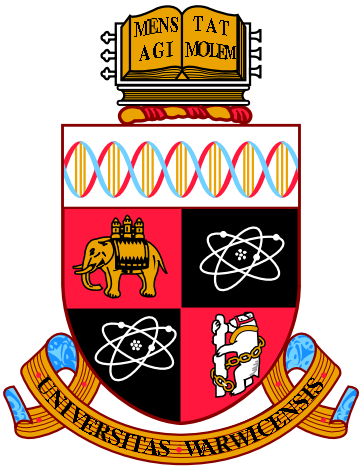


Features Correlation-based Workflows for High-Performance Computing Systems Diagnosis

by

Thuan Yew Edward Chuah

A thesis submitted to The University of Warwick

in partial fulfilment of the requirements

for admission to the degree of

Doctor of Philosophy in Computer Science

Department of Computer Science

The University of Warwick

June 2020

Contents

List of Tables

List of Figures

Abstract

Analysing failures to improve the reliability of high performance computing systems and data centres is important. The primary source of information for diagnosing system failures is the system logs and it is widely known that finding the cause of a system failure using only system logs is incomplete. Resource utilisation data – recently made available – is another potential useful source of information for failure analysis. However, large High-Performance Computing (HPC) systems generate a lot of data. Processing the huge amount of data presents a significant challenge for online failure diagnosis. Most of the work on failure diagnosis have studied errors that lead to system failures only, but there is little work that study errors which lead to a system failure or recovery on real data.

In this thesis, we design, implement and evaluate two failure diagnostics frameworks. We name the frameworks CORRMEXT and EXERMEST. We implement the Data Type Extraction, Feature Extraction, Correlation and Time-bin Extraction modules. CORRMEXT integrates the Data Type Extraction, Correlation and Time-bin Extraction modules. It identifies error cases that occur frequently and reports the success and failure of error recovery protocols. EXERMEST integrates the Feature Extraction and Correlation modules. It extracts significant errors and resource use counters and identifies error cases that are rare. We apply the diagnostics frameworks on the resource use data and system logs on three HPC systems operated by the Texas Advanced Computing Center (TACC). Our results show that: (i) multiple correlation methods are required for identifying more dates of groups of correlated resource use counters and groups of correlated errors, (ii) the earliest hour of change in system behaviour can only be identified by using the correlated resource use counters and correlated errors, (iii) multiple feature extraction methods are required for identifying the rare error cases, and (iv) time-bins of multiple granularities are necessary for identifying the rare error cases. CORRMEXT and EXERMEST are available on the public domain for supporting system administrators in failure diagnosis.

Acknowledgements

I would like to thank the academics, engineers and administrators who provided valuable support to my work. Firstly, I would like to thank my PhD supervisor, Dr. Arshad Jhumka. He and I have collaborated on research in failure diagnosis since 2012. He is open to new angles of research in fault tolerance and engages in interesting discussion on research. Because of that, I can work on a research topic that I am really interested in and collaborate with him. From 2013 – 2019, we published 10 papers in peer-reviewed journal, conference and workshop proceedings. I am a first author on eight papers in system failure diagnosis. Thank you, Arshad.

I would like to thank my industry collaborators at Intel Corporation. I would like to express my heartfelt appreciation to Mr. Karl Solchenbach. During my PhD, he introduced his engineers, Dr. Samantha Alt, Dr. Marie-Christine Sawley and Dr. Joshua B. Fryman. They provided constructive feedback that improved my papers significantly and suggested ideas to make my PhD work more interesting to the industry. It led to the successful publication of my conference paper in 2017 and two more papers in 2019. Thank you, Karl, Samantha, Marie-Christine and Joshua.

I would like to thank my collaborators at the Rutgers Discovery Informatics Institute at Rutgers University, USA. I would like to express my gratitude to Professor Manish Parashar. When I asked for his help, he introduced Dr. Daniel Balouek-Thomert and Dr. J.J. Villalobos from his team who guided me on my papers. Dr. Thomert, Dr. Villalobos and Prof. Parashar not only helped me improve the clarity of my writing and presentation of the results, they also helped me understand what the anonymous reviewers were asking and showed me how to address the comments. Because of them, I can address all the concerns that were raised by the anonymous reviewers. It led to the successful publication of my journal and conference papers

in 2019. Thank you so much, Daniel, Juanjo and Prof. Parashar.

I would like to thank Dr. William L. Barth, Dr. Tommy Minyard and Dr. Richard T. Evans at the Texas Advanced Computing Center at The University of Texas at Austin. Bill, Tommy and Todd granted me access to their HPC systems and provided the important ingredient for my PhD: the system logs and resource utilisation data. Without their data, my research would be more difficult to do. Thank you, Bill, Tommy and Todd.

I would like to thank my advisors Dr. Victor Sanchez and Dr. He Ligang at the Computer Science department at The University of Warwick for their helpful comments at my annual review. I would also like to thank Dr. Theo Damoulas (University of Warwick) for introducing the Bonferroni Correction method.

I would like to thank my examiners Dr. He Ligang and Professor Miroslaw Malek. During my Viva, Dr. He and Professor Malek not only provided constructive feedback which helped improve my final thesis significantly, they also offered new research directions to extend the work I did during my PhD studies. Because of them, I can pursue my interest in the field of fault tolerance using statistics and data analytics for years to come. Thank you so much, Ligang and Prof. Malek.

I would like to thank the teams from Student services and Data Science at Scale programme at The Alan Turing Institute for the strong administrative support they have given to me during my PhD. Specially, I would like to mention Emma Armitage, Samantha Selvaraj, Dr. Ben Murton, Dr. Donna Brown, Georgia Koumara, Emma Levy, Dr. Helen Davies, Catherine Lawrence, Anastasia Shteyn, Dr. Nico Guernion and Dr. Anthony Lee. I would also like to thank Sharon Howard and Ruth Cooper for the strong administrative support they gave to me at the Computer Science department at The University of Warwick. Without their support, I would not be able to complete my thesis on time. Thank you all for the support.

Dr. James Clayton Browne (January 16, 1935–January 19, 2018) was Regents Chair in Computer Sciences, Professor of Physics and Professor of Electrical and Computer Engineering at The University of Texas at Austin. Professor Browne's contributions to the development of the failure diagnosis frameworks have been crucial but unfortunately he passed away when I was working on my PhD. In 2016,

I almost gave up the PhD in Computer Science specialising in fault tolerance but he inspired me with his positive manner, motivated me to carry on, provided me with strong reference letters and supported me with the limited resources he had. As such, I dedicate this thesis to the memory of Dr. James Clayton Browne.

Dedication

In memory of Dr. James Clayton Browne – Mentor and teacher.

"*For God so loved the world that He gave His only begotten Son, that if he believes in Him he will not perish but have everlasting life.*" – John 3:16.

Declarations

Parts of this thesis have been previously published by the author in the following:

[10] Edward Chuah, Arshad Jhumka, Samantha Alt, Theo Damoulas, Nentawe Gurumdimma, Marie-Christine Sawley, William L. Barth, Tommy Minyard, and James C. Browne. Enabling dependability-driven resource use and message-log analysis for cluster system diagnosis. In *Proceedings of IEEE International Conference on High Performance Computing, Data and Analytics (HiPC)*, pages 317–327, 2017. doi: 10.1109/HiPC.2017.00044

[11] Edward Chuah, Arshad Jhumka, Samantha Alt, Daniel Balouek-Thomert, James C. Browne, and Manish Parashar. Towards comprehensive dependability-driven resource use and message log-analysis for HPC systems diagnosis. *Journal of Parallel and Distributed Computing*, 132:95–112, 2019. doi: https://doi.org/10.1016/j.jpdc.2019.05.013

[12] Edward Chuah, Arshad Jhumka, Samantha Alt, J.J Villalobos, Joshua B. Fryman, William L. Barth, and Manish Parashar. Using resource use data and system logs for HPC system error propagation and recovery diagnosis. In *Proceedings of IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 1–10, 2019

Research was performed in collaboration during the development of this thesis, but does not form part of the thesis:

[9] Edward Chuah, Arshad Jhumka, James C. Browne, Nentawe Gurumdimma, Sai Narasimharmuthy, and Bill Barth. Using message logs and resource use data for cluster failure diagnosis. In *Proceedings of IEEE International Conference*

on High Performance Computing, Data and Analytics (HiPC), pages 232–241, 2016. doi: 10.1109/HiPC.2016.035

[6] Edward Chuah and et. al. Enabling online resource use and system log-analysis for HPC systems vulnerability diagnosis. In *Under preparation for submission in 2020*, 2020

Sponsorships and Grants

This research was supported by The Alan Turing Institute under the EPSRC, UK grant EP/N510129/1, The Alan Turing Institute–Intel partnership, The University of Warwick Department of Computer Science scholarship and the National Science Foundation, USA under OCI awards #0622780, #1203604 and #1134872 to the Texas Advanced Computing Center at The University of Texas at Austin.

Acronyms

AUC Area Under Curve.

CORRMEXT **COR**relating **R**esource use data and **M**essage logs and **EX**tracting **T**imes.

CPU Central Processing Unit.

CRC Cyclic Redundancy Check.

DBSCAN Density-Based Spatial Clustering of Applications with Noise.

DILAF DIstributed Log Analysis Framework.

DOC Decentralized Online Clustering.

ECC Error Correcting Code.

EXERMEST **EX**tracting F**E**atures and Co**R**relating Resource Use Counters and **MES**sage **T**ypes.

FSA Finite State Automaton.

GB Giga Byte.

GHz Giga Hertz.

GPF General Protection Fault.

HDFS Hadoop Distributed File System.

HELO Hierarchical Event Log Organizer.

HPC High Performance Computing.

I/O Input Output.

IBM International Business Machines.

ICA Independant Component Analysis.

IPLoM Iterative Partitioning Log Mining.

LKE An unstructured log-analysis technique for anomaly detection.

LogAider A tool for mining event correlations in HPC system event logs.

LoGS A log-analysis tool that is implemented to make the administration of large clusters more bearable.

LogSig A message based signature algorithm that generates system events from raw textual message logs.

Logsurfer A log-file analysis tool.

MB Mega Byte.

Message Type A sequence of words where each word contains only alphabets in the English language.

MLCS Multiple Longest Common Subspace.

MPRIME Mersenne prime number test application.

MRPC Most Relevant Principal Component.

MTE Message Template Extractor.

MTExt Message Types Extractor.

NLPCA Non-Linear Principal Component Analysis.

NUMA Non-Uniform Memory Access.

PCA Principal Component Analysis.

RAS Reliability, Serviceability and Availability.

RUExt Resource Use Extractor.

SEC Simple Event Correlator.

SIG Structure-of-Influence Graph.

SLCT Simple Log Clustering Tool.

SVM Support Vector Machine.

SWATCH A configurable log file filter and monitor.

TACC Texas Advanced Computing Centre.

TF/IDF Term Frequency/Inverse Document Frequency.

Time-bin A time-bin is a time window of one fixed time interval.

Chapter 1

Introduction

The supercomputing and dependable systems communities have shown increasing interest in failure analysis for data centres and high-performance computing (HPC) systems; demonstrated from publications that report large-scale analysis of hardware failures in data centres [68] and long-term measurement and analysis of failures in HPC systems [30]. There is a lot of research on detecting failure-inducing errors and analysing and diagnosing system failures. A significant body of methods have shown the value of system logs for detecting errors [50, 69], predicting failures [14, 18, 22, 46, 52] and diagnosing failures [41, 51, 55]. However, the message structure is known to vary widely. By vary widely, we mean that both the text and alpha-numerical words in the message appear in no specific order. The system logs are redundant and provide low coverage. By low coverage, we mean that the system logs do not contain all the events needed to establish a complete causal trace path to the failure. By redundant, we mean that only a small quantity of the system logs is relevant to the diagnosis of a failure. Another significant body of methods have shown the value of resource use data for detecting anomalies [28], tracking system behaviour [61], characterising errors [5, 29], and predicting system failures [58] or system recovery [31]. The methods show that using resource use data and system logs *separately* can help system administrators manage the complexity of a HPC system and data centre.

1.1 Motivation

Recent work that combine system logs and resource use data for detecting errors [16, 33, 34] and diagnosing system failures [8, 72] have shown increased accuracy over the use of system logs alone. The work by N. Gurumdimma et. al. [33, 34] show that the time window for handling errors and the accuracy for detecting errors can be increased by combining resource usage data and system logs. The work by Z. Zheng

et. al. [71] combines RAS logs (Reliability, Serviceability and Availability) and Job logs in a large scale cluster system for identifying interesting failure characteristics. The ANCOR framework [8] combines resource usage data and system logs in a two-phase approach for diagnosing system failures. The first phase identifies system usage anomalies in the resource use data to provide a partial diagnosis of node failures. Then in the second phase, system message log-analysis is provided to give a more detailed diagnosis. The work by S. Di et. al. [16] identifies correlations across nodes and time in a HPC system by combining RAS logs and Job logs for detecting errors that lead to a system failure. The error detection and failure diagnostics frameworks presented in references [8, 16, 33, 34, 72] have identified system errors that lead to a system failure only. However, there is little work that study errors which lead to a system failure or recovery. Knowing when an error leads to a success or failure of an error recovery protocol can be useful to system designers and implementors; the knowledge can be used to improve the effectiveness of error recovery protocols.

1.2 Background

In this section, we explain fault tolerance, give the basic definition of an error and a failure, describe failure diagnosis and give an example of a diagnostics scenario.

1.2.1 Fault Tolerance

A summary of fault tolerance techniques is shown in Figure 1.1.

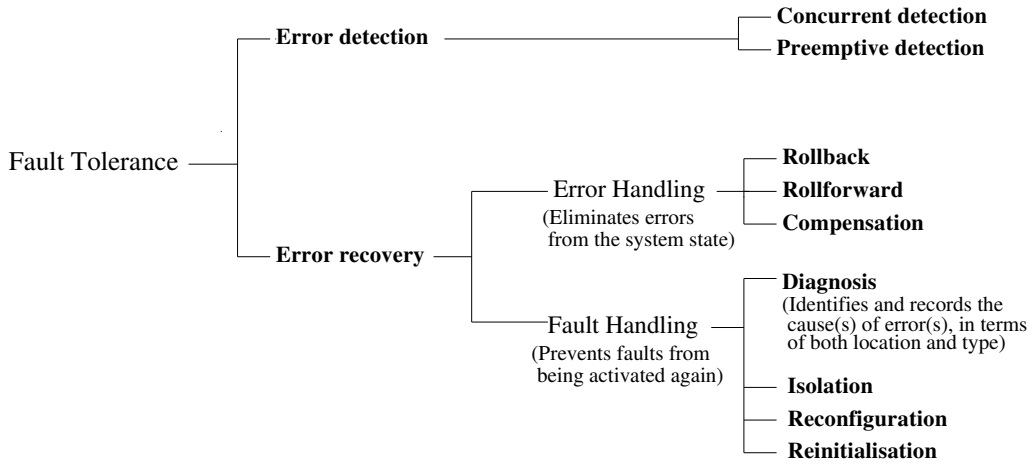


Figure 1.1: A summary of fault tolerance techniques. The figure was modified using Fig. 16 in the reference [3].

Fault tolerance is a property that enables a computer system to continue operating properly in the event of a failure of some of its components. If the operating

quality of the computer system decreases, the decrease is proportional to the severity of the failure. A computer system that is designed to be fault tolerant can continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails. Within the scope of a HPC system or data centre, fault tolerance can be achieved by anticipating exceptional conditions and building the system to cope with them.

1.2.2 Errors and Failures

An *error* is defined as the part of the total state of the system that may lead to its subsequent failure [3]. An error is detected if its presence in indicated by an error message or error signal. An error can also be present but not detected; these are called *latent* errors. A *failure* is defined as an event that occurs when the state of the system deviates from the expected one. Some errors do not lead to a system failure but some errors do reach the external state of the system and cause a failure. A system failure occurs when an error propagates from one component to another until it has reached the system boundary. Errors can propagate to the system boundary if error recovery mechanisms fail or there is a lack of recovery mechanisms.

1.2.3 Failure Diagnosis

Failure diagnosis is a process of tracing a fault, i.e., the cause of a failure by means of its symptoms, applying knowledge and analysing test results [25]. Error detection assumes prior knowledge about the fault model. Differently to error detection, failure diagnosis assumes no prior knowledge about fault models. Accurate diagnosis of failures in HPC systems and data centres requires: (i) collecting information from sensors, (ii) processing the information using advanced data preprocessing techniques, and (iii) extracting the required features for efficient identification of faults. When the cause of a failure is identified, remedial actions can be taken to increase productivity and reduce maintenance in the HPC system and data centre. Machine learning methods (e.g., feature extraction) and statistical techniques can be used for developing tools which can aid the system administrators in their day-to-day task of managing a complex HPC system or data centre.

Error detection: Error detection is the detection of errors caused by noise or other impairments introduced into data while it is transmitted from source to destination. When an error message or error signal is indicated, the error is detected. When an error is present but not detected, the error is called a latent error. Error recovery is the ability to resume operation after encountering the error. The ability to detect an error and reconstruct the original error-free state is called error correction. This thesis focuses on HPC systems diagnosis.

1.2.4 A Diagnosis Scenario

A diagnosis scenario between an application and a remote file-system is illustrated in Figure 1.2.

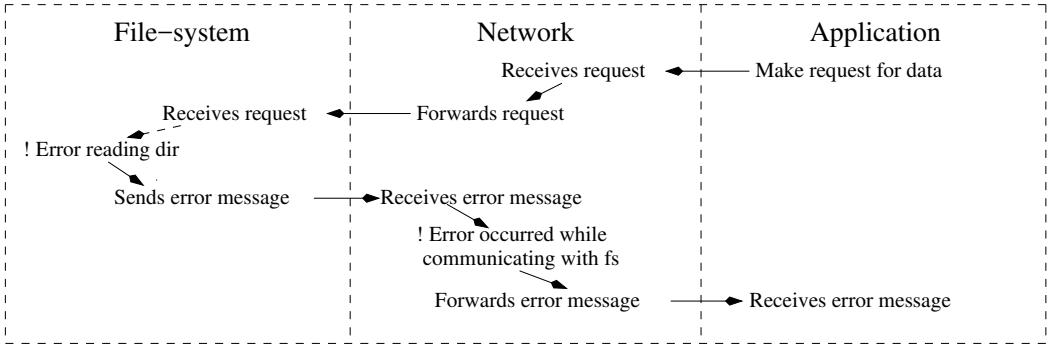


Figure 1.2: An illustration of errors propagating between the application, network and file-system.

When an application sends a request for data that is stored on a networked filesystem, the filesystem will respond to the client. In most cases, the filesystem will retrieve the data and send the data to the client. However, in some cases the filesystem may encounter a directory read error. It is unable to access the directory to read the data. The filesystem will generate an error message and send it to the client. Then, the client will send another request to the filesystem. There are many reasons for read errors to occur on a filesystem. Possible causes for filesystem read errors include bad blocks on the harddisk or another client has obtained a lock on the directory that holds the data. Some filesystem read errors are recoverable. If the read error is caused by an application holding a lock on a directory, when the lock is released, the filesystem is able to access the directory, read the data and send it to the client. If the read error is caused by bad blocks on the harddisk, then the data stored on the disk is lost.

1.3 The Problem

Modern day data centres and HPC systems suffer from occurrence of errors and failures. The data centres and HPC systems are comprised of different combinations of processors, networks, operating system processes, memory and storage systems. When new technologies and software deploying in HPC systems and data centres grow, the nature of computing systems can change rapidly. The computing systems are capable of generating a lot of data and different types of data. However, the data are not necessarily structured. Therefore, it is a challenging task to find the

right types of data and analyse the data to quickly diagnose system problems. For the system administrators, managing these complex computing systems is a labour intensive day-to-day task. Therefore, it is important to address the challenge by providing tools that can help the system administrators identify where, when and why the error recovery protocol failed or succeeded. In this thesis, we are interested in capturing the frequent error cases and the rare error cases.

1.4 The Approach

In this thesis, we design, implement and evaluate two system diagnostics workflows as shown in Figure 1.3.

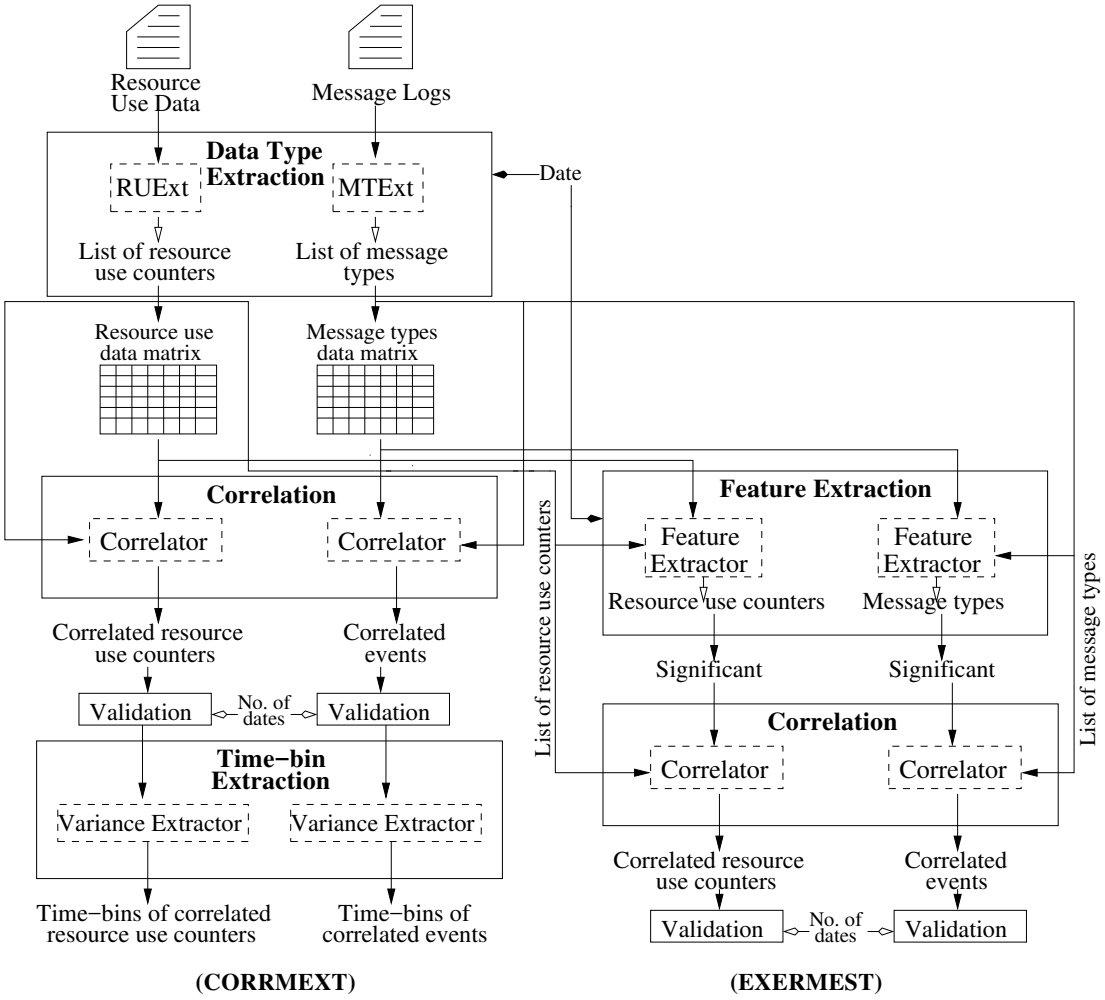


Figure 1.3: The workflows of the diagnostics approach. The workflow for the CORRMEXT framework is shown on the left. The workflow for the EXERMEST framework is shown on the right. Each module produces a report which can be used for diagnosis.

The first diagnostics framework – we called CORRMEXT (**COR**relating **R**esource use data and **M**essage logs and **EX**tracting **T**imes) – integrates data type extraction, correlation and time-bin extraction methods. It identifies frequently occurring error cases. CORRMEXT uses both TACC_Stats [17] resource usage data and system logs in its analyses. The second diagnostics framework – we called EXERMEST (**EX**tracting F**E**atures and Co**R**relating Resource Use Counters and **MES**sage **T**ypes) – integrates feature extraction and correlation methods. Differently to CORRMEXT, EXERMEST identifies rare error cases. CORRMEXT can be used to identify error cases that occur frequently. EXERMEST can be used to identify error cases that are rare.

1.5 Contributions of this Thesis

The techniques for achieving fault tolerant HPC systems and data centres are presented in Figure 1.1. Failure diagnosis – a fault removal technique – is one way to achieve a fault tolerant HPC system. There are many frameworks that identify errors that lead to a system failure, but there is little work that study errors which lead to a system failure or recovery on real data. Existing failure diagnosis techniques correlate a given error event (for example, a file-access error) to a system failure and return the most likely cause of the failure. While information like these can provide some details on the likely cause of the failure, capturing the propagation paths of errors and recovery attempts provide more details. A diagnosis such as *"A DNS lookup failure was caused by a DNS software configuration error. The configuration error actually resulted in the DNS lookup failure which led to the FTP software failure"* provides more details than *"A DNS lookup failure caused the FTP software failure"*. In the former, the diagnosis tells the system administrator that the fault lies in the DNS software configuration. Therefore, they can fix the problem by configuring the DNS software correctly. In this thesis, we design, implement and evaluate two new workflows for system diagnosis. We obtain the resource use data and system logs on three different HPC systems and conducted extensive experiments to validate the workflows. Our results show that: (i) we require multiple correlation methods to identify more dates of groups of correlated resource use counters and groups of correlated errors, (ii) we require both the correlated resource use counters and correlated errors to identify the earliest hour of change in system behaviour, (iii) we require multiple feature extraction methods to identify the rare error cases, and (iv) we require multiple granularities of time-bins to identify the rare error cases.

- We design, implement and evaluate a new failure diagnostics framework – called CORRMEXT – that combines analyses of resource use data and system

logs for detailed dependability-oriented system failure diagnosis. CORRMEXT evaluates multiple correlation algorithms. We show that: (i) multiple correlation algorithms are required to identify more dates of groups of correlated resource use counters and correlated errors, (ii) both the correlated resource use counters and correlated errors are required for identifying the earliest hour of change in system behaviour. We study different error cases on the Ranger HPC system. We report this contribution in Chapter 4 of this thesis.

- We show that the CORRMEXT framework generalises to multiple HPC systems. We: (i) confirm that more dates of groups of correlated resource use counters and groups of correlated errors can only be identified by applying multiple correlation algorithms, and (ii) confirm that the earliest times of change in system behaviour can only be identified by analysing both the correlated resource use counters and correlated errors. We study different error cases on Lonestar4 and Stampede-1. We report this contribution in Chapter 5 of this thesis.

- We develop a new workflow for identifying rare error cases on multiple HPC systems. The workflow is implemented in a framework called EXERMEST. The EXERMEST framework evaluates three feature extraction algorithms and shows that we require multiple feature extractors to identify the significant resource use counters and errors associated with the rare error cases. By significant, we mean the highest scores that the feature extractors assign to the resource use counters and errors. We study different error cases on Ranger and Lonestar4. We report this contribution in Chapter 6 of this thesis.

1.6 Outline of this Thesis

The remainder of this thesis is structured as follows:
Chapter 2 presents a detailed survey of system log-file processing tools. It describes the main techniques developed by the tools. The techniques are: (i) regular expressions, (ii) clustering, (iii) feature selection, (iv) anomaly detection, and (v) hybrid method. A hybrid method uses two or more techniques. The chapter highlights the similarities and differences between the system log-file processing tools.
Chapter 3 presents the problems that we solve in this thesis. It introduces the system model, fault model and system issues. It describes: (i) Ranger, Lonestar4 and Stampede-1 HPC systems operated by the Texas Advanced Computing Center at The University of Texas at Austin, (ii) cluster log-data generated on these HPC systems, and (iii) implementation details for the data preprocessing modules. Parts

of the chapter use material in the conference paper [10] and journal paper [11]. Specifically, it uses:

- the system model reported in [8], the fault model and data preprocessing modules reported in [10].
- the system issue reported in [10, 11].

Chapter 4 presents CORRMEXT to identify frequently occurring error cases and report the rates of success and failure of error recovery protocols. It describes the data type extraction, correlation and time-bin extraction modules and presents five different error cases identified on the Ranger HPC system. Parts of the chapter use material in the conference paper [10] and journal paper [11]. Specifically, it uses the problem specification, data type extraction, correlation and time-bin extraction modules, and five error cases identified on the Ranger HPC system.

Chapter 5 shows that CORRMEXT generalises to multiple HPC systems. It presents several new findings and failure patterns, gives the details of two different error cases on Lonestar4 and one error case on Stampede-1, and describe the benefit of combining analysis of resource use data and system logs for HPC systems diagnosis. Parts of the chapter use material in the journal paper [11]. Specifically, it uses the two error cases reported in [11].

Chapter 6 presents a features correlation-based workflow called EXERMEST. It describes the feature extraction and correlation modules in the framework and presents three rare error cases identified on Ranger and one rare error case on Lonestar4. Parts of the chapter use material in the conference paper [12]. Specifically, it uses the problem specification, feature extraction and correlation modules, and three error cases reported in [12].

Chapter 7 presents a comparative analysis of CORRMEXT and EXERMEST. It discusses the functions and types of errors diagnosed by CORRMEXT and EXERMEST. We refer the reader to this chapter for comparing the diagnostics capabilities of CORRMEXT and EXERMEST.

Chapter 8 summarises this thesis and provides a number of suggestions for future research.

Chapter 2

Literature Review

When a system failure occurs, the first place that the system administrator goes to to identify the cause of a failure are the system logs. The system logs contain streams of system events generated by the operating system, filesystem and system software. The system events are interleaved in time and only a small fraction of the events are relevant to the diagnosis of a given failure. In this chapter, we present a detailed survey of system log-file processing tools.

We structure this chapter according to the main techniques developed by the tool. The techniques are: (i) regular expressions (Section 2.1), (ii) clustering (Section 2.2), (iii) feature selection (Section 2.3), (iv) anomaly detection (Section 2.4), and (v) hybrid method (Section 2.5). In Section 2.6, we highlight the similarities and differences between the system log-file processing tools and conclude with a summary in Section 2.7.

2.1 Regular Expressions

In this section, we present the tools that develop regular expressions as the main technique for processing textual system event logs. The tools are LoGs [54], Simple Event Correlator [56, 65], SWATCH [37] and LogSurfer [53].

2.1.1 LoGS

LoGS [54] is a log-analysis tool that is implemented to make the administration of large clusters more bearable. It has a log-analysis engine which consists of five different components. The components of the log-analysis engine are: messages, rules, rulesets, actions and contexts. Messages can be a single line from a syslog file. Rules match messages and trigger actions based upon those matches. A rule consist of seven optional components: match function, no-match function, delete-rule function,

no-delete-rule function, absolute timeout, continue value and a list of actions. A ruleset shares many properties of rules and a ruleset can match messages. When a ruleset matches a message, the rules inside a ruleset are checked against the message. This functionality allows a ruleset to be organised in a tree and it reduces the number of comparisons that must be made to determine if a message is interesting. An action define what is to be done with a message or context of messages. An action can be triggered when a rule matches a message, a ruleset matches a message or a context exceeding its limits. Actions are Common Lisp functions. Contextual information are used by LoGS to gather and store related messages for potential use. Related messages may be analysed as a group.

2.1.2 SEC

The Simple Event Correlator (SEC) [56, 65] is a real-time log-file analysis tool that is developed to provide support for recognising the temporal component of log analysis. SEC receives input events from a file-stream and produces output events by executing user-specified shell commands. The input events include regular files, named pipes and standard input. The input events are handled as regular expressions and configuration files that contain rules can be created and stored.

The rule types that are supported include *Single*, *SingleWithScript*, *SingleWithSuppress*, *Pair*, *PairWithWindow*, *SingleWithThreshold*, *SingleWith2Threshold*, *Suppress* and *Calendar*. The *Single* rule type matches an input event and executes a list of actions. The *SingleWithScript* rule type matches an input event and executes a list of actions if an external script or program returns certain exit values. The *SingleWithSuppress* matches an input event and executes a list of actions but it will ignore the matching events which follow for a specified number of seconds. The rule *Pair* matches an input event and executes a list of actions, ignore the matching events which follow for a specified number of seconds and then match a second event and execute another list of actions. The rule *PairWithWindow* executes the rule type *Pair* within a given time-window. The rule type *SingleWithThreshold* counts matching input events within a given time-window and executes a list of actions if a given threshold has exceeded. The rule type *SingleWith2Thresholds* executes the rule type *SingleWithThreshold* two times within two time-windows and two threshold values. The rule type *Suppress* suppresses a matching input event and the rule type *Calendar* executes a list of actions at specified times.

SEC supports a comprehensive list of actions and the list of actions include log a message, write a line to a file, execute an external shell script or program, and generate a new input event that can be matched by other rules.

2.1.3 SWATCH

SWATCH [37] is a configurable log file filter and monitor that monitors log files, filters out unwanted data and takes user-specified actions based upon the patterns in the log. SWATCH can be used in three different ways. The first way is to make a single pass through a file. The second way is to look at messages that are appended to a file as the file is being updated. The third way is to examine the standard output of a program. It has three basic parts. The first part is a configuration file. The second part is a library of actions and the third part is a controlling program. A non-comment line in a configuration file is made up of a pattern expression, a list of actions to be performed if the expression is matched, an optional time interval and the location of a timestamp.

The list of actions SWATCH understands include echo, bell, ignore, write and mail, and pipe and exec. The action *echo* echos the line to SWATCH's controlling terminal. The action *bell* sends a bell signal to the controlling terminal. The action *ignore* ignores the current line of input and proceeds to the next line. The actions *write* and *mail* sends a copy of the line to the user via the write and mail commands. The action *pipe* matches lines as input to a particular command on the system. The action *exec* runs a command on the system with the option of using selected fields from the matched lines as arguments to the command.

2.1.4 Logsurfer

Logsurfer [53] is a log-file analysis tool that examines messages in a log file in terms of how they relate to other messages. There are four different components in Logsurfer. The components are messages, rules, actions, and contexts. Messages are pieces of incoming data from a single data source. Rules match messages and trigger actions based on those matches. Actions are triggered when a rule matches a message or when a context is closed. Actions can also create new rules, create new contexts and delete unneeded contexts. Contexts gather and store messages in memory. A logsurfer rule specification is made up of seven fields. The first field specifies a regular expression that indicates which messages the rule is matched. The second field specifies a regular expression that indicates exceptions to the first field. The third field specifies a regular expression that indicates when a rule is to be deleted. The fourth field specifies a regular expression that indicate exceptions to the third field. The fifth field specifies the amount of time in seconds a rule can be active. The sixth field can be used to indicate when a rule matches a message, subsequent rules should also try to match the message. The seventh field specifies an action to be taken when a rule matches a message.

Seven action types are supported by Logsurfer. The action *ignore* disregards a message and it is often used to filter out messages that need not be acted upon. The action *exec* executes an external program. The action *pipe* executes an external program with the current message or context as its standard input. The action *open* opens a new context. The action *delete* deletes a context. The action *rule* creates a new rule.

A single message is often not enough to tell whether there is a problem. Logsurfer employs contexts that collect and store messages in memory and it uses a pair of regular expressions to determine which messages are added to a context. The first regular expression in the pair specifies the messages to add to the context, and the second regular expression specifies exceptions to the first regular expression.

2.2 Clustering

In this section, we present the tools that develop clustering as the main technique for processing the textual system event logs. The tools are SLCT [66], IPLoM [48], HELO [23], Baler [49], Decentralized Online Clustering [52] and Handling redundancy [32].

2.2.1 SLCT and LogHound

SLCT (Simple Log Clustering Tool) [66] is a data clustering algorithm that is developed for analysing textual event logs. Each log line in the event log is viewed as a data point with categorical (non-numeric) attributes. SLCT groups data points into clusters such that the data points in a cluster are similar to each other, and data points that do not fit in any cluster are categorised as outliers. SLCT uses a density-based algorithm that identifies dense regions in the data space and forms clusters from them. The author also discussed about the traditional distance-based clustering approach. SLCT does not use the distance-based clustering approach because defining a distance function for categorical data is not trivial and the notion of distance becomes meaningless in a high-dimensional data space. LogHound uses a frequent itemset mining algorithm to discover frequent patterns in event logs.

2.2.2 IPLoM

The reference in [48] presents a novel algorithm called IPLoM (Iterative Partitioning Log Mining) for mining clusters in event logs. IPLoM finds frequent event type patterns and works through a 3-step hierarchical clustering process. The hierarchical clustering process partitions the event logs by token count, token position and search for bijection. A token is a sequence of characters with no space. Step 1 of

the algorithm creates partitions of log messages where each partition contains log messages that have the same token length. Step 2 of the algorithm finds the position of a token which has the least unique values in a partition and uses the token position to split the partition. Step 3 of the algorithm finds 1-to-1 relationships between tokens in a partition and splits the log messages that contain these tokens into a separate partition. The final step in the algorithm discovers cluster descriptions in each partition. If a partition contains one unique token, the token name is used to describe the cluster. On the other hand, if a partition contains two or more unique tokens, a wildcard is used to describe the cluster. The authors compared IPLoM with SLCT, LogHound and Teiresias. IPLoM achieved an average F-measure of 78% when the closest other algorithm achieved an F-measure of 10%.

2.2.3 Helo

HELO [23] is a novel unsupervised clustering engine that accurately mines event type patterns from log-files generated by large supercomputers. It has two different parts: Offline classification and online classification. The first part, offline classification, finds events in log-files and uses the events to create the first template set. The algorithm uses a 2-step hierarchical process to group events according to their message description. In the first step of the process, the best split column for each cluster is searched. In the second step of the process, the clusters are divided. A split column represents a word position in the message description and the word position in the message is used to divide the cluster into different groups.

The second part, online classification, groups messages as they are being generated by the system. The input to the online classification part is given by the groups obtained from the offline process on the initial dataset. On receiving a new message, the online component checks the description of the message and finds the most appropriate group templates. A new message is added to a group if it fits the exact description of a group. A new message that does not have an exact match with any of the groups is added to a group if the message does not decrease the cluster goodness under a threshold. If the cluster goodness of all groups decrease, a new group is formed and the message is added to the group. The authors compared HELO with five tools. The tools are IPLoM, StrApp, LogHound, SLCT and MTE. HELO was shown to outperform the tools on precision, recall and F-measure.

2.2.4 Baler

Baler [49] is a deterministic, lossless log message clustering tool that is developed to handle large datasets, the quality of result patterns and explore the results. The

Baler log-clustering engine has two processing phases: An initial processing phase and a clustering phase. It tokenises the data and performs clustering based on the token attributes. The methodology processes the input data using only a single pass and processes new log-files incrementally. In the initial processing phase, the input message is parsed using a grammar that comprises of five non-terminals. The non-terminals are *alpha-numeric*, *white space*, *colon*, *bracket* and *other symbol*. Each of the non-terminals is described using regular expressions. The input message is then tokenised into a sequence of symbol tokens, with each token having a token-id. The token attributes are token ID, token type, string value and integer value. When a token is parsed, its string value is checked against a mapping for an existing token. If the token exists, its ID is pushed into a sequence of integers that represent the message. If the token does not exist, a new token is created and a new ID and token attributes are assigned. The new token ID is inserted into the sequence of integers. Nine token types are defined. The token types are *ENGLISH*, *NUMBER*, *ALPHA-NUMERIC*, *SPACE*, *COLON*, *BRACKET*, *OTHER*, *SEQ* and *KLEEN_STAR*. The token type *SEQ* is a sequence of tokens. The token type *KLEEN_STAR* is a container of tokens. The token type *OTHER* is other symbol.

The log-clustering engine uses English terms to determine message patterns. It uses a heuristic to extract patterns from a parsed log message. A pattern is inserted into the pattern hash, where the pattern key is a sequence of ids of the pattern tokens. In the clustering phase, the similarity between pairs of 1-patterns is evaluated. Levinstein distance measure is used to infer the similarity between pattern tokens. Two criteria that assess the similarity are defined. In the first criteria, two 1-patterns are considered to be similar when the number of changes is less than a given threshold. In the second criteria, the ratio of the changes count and the minimal length of the 1-patterns is used to determine the similarity.

The authors compared Baler with Teiresias, SLCT, Loghound and IPLoM on three conditions of interest that occurred in a cluster system at Sandia National Laboratories. The conditions of interests are *CPU stuck*, *oom-killer* and *EDAC memory error*. Baler identified more patterns for the conditions of interests than Teiresias, SLCT, Loghound and IPLoM.

2.2.5 Decentralized Online Clustering (DOC)

A. Pelaez et. al. [52] present a solution for predicting compute node soft lockups via online anomaly detection using a decentralised online clustering algorithm. Resource anomalies in resource usage logs have been found in faulty nodes and reported in the reference [8]; this means that detecting anomalies in node resource usage at runtime can be used for predicting compute node soft lockups. The Decentralized

Online Clustering algorithm has been developed for distributed system monitoring. The rationalized message logs collected on the Ranger supercomputer at the Texas Advanced Computing Center is used for evaluating the effectiveness of the solution.

The solution has three main components. The first component, called DOC, is responsible for distributing points to the nodes and implements the clustering algorithm. The points are the resource usage features. The second component extracts anomalies from the clusters generated by DOC. It also keeps track of the logs and extracts the relevant features from the logs for invoking DOC with the correct data. The third component makes predictions based on the anomalies identified in the second component. It also uses multiple time bins and multiple clustering for improving the accuracy of the predictions.

DOC is designed to run in a distributed setting on data that is also distributed. The data is usually generated on the nodes and then the data is input into DOC. DOC applies a density-based clustering approach. Clusters are detected by evaluating the relative density of points within the information space. The information space is subdivided dynamically into regions and each region is assigned to a particular processing node using a content-based Distributed Hash Table that uses a Peano-Hilbert space-filling curve. The Peano-Hilbert space-filling curve gives a mapping between 1-Dimensional and 2-Dimensional space that preserves locality and is used to assign processing nodes to regions. The Distributed Hash Table is used to get resource usage features to the processing nodes within each region in a scalable way. DOC is designed to be flexible to changes in topology. Because the topology of cluster systems is stable, distributing the resource usage features and performing cluster data aggregation is sped up by adding a caching layer to DOC. The range of addresses that a node is responsible for is returned when a lookup operation for a resource usage feature in the distributed hash table is resolved. The address range and the node identifier are saved in the local cache. When another point that belongs to the same address range is to be inserted, no lookup will be performed; the insertion request is sent directly to the node found in the cache which reduces the communication overhead required during each lookup. The resource usage feature is inserted only if it belongs to the node's information space region; this guarantees the coherence of the cache. Two complementary strategies are developed for improving the precision of the approach. In the first strategy, multiple time bins - a time bin is an interval of time where resource usage data is collected - were experimented. From the experiments, it was observed that pairs of five and ten minute time bins increased the precision of the approach. In the second strategy, multiple clustering was performed on different subsets of features including the original feature set. It was observed that outliers present in multiple clustering runs are more correlated to

soft lockups.

The authors used two dimensions for testing the usefulness of the approach. The two dimensions are accuracy of the prediction and scalability and performance. In the first dimension, the rationalized message logs for the first week of March 2012 was collected and the solution is evaluated against PCA and DBSCAN using the collected logs. The results show that the approach outperformed PCA and DBSCAN in terms of job precision by 0.2 and node precision by 0.47. In the second dimension, the Stampede supercomputer was used to perform all the runs of the solution and the time measurements were split into two parts. The first part measured the time DOC took to redistribute all the resource usage features to the processing nodes. The second part measured the time DOC took to find all the outliers and report the results back to the processing node. The results show that the clustering time accounted for only 9.5% of the total time. For 5k nodes, the total time that the solution took is just slightly more than three minutes.

2.2.6 Handling redundancy

N. Gurumdimma et. al. [32] develop a novel generic log compression technique for addressing the problem of redundant messages in cluster system logs. The technique uses iterative clustering. It computes the Leveinstein distance metric to measure the difference between two strings. The technique has two steps: Tokenisation and parsing, and filtering. In the first step, the logs are parsed to obtain the event types and event attributes. Tokens containing alpha-numeric words and English-only words are kept while tokens containing only numbers are removed. In the second step, the distance between tokens is measured using the Leveinstein distance metric. The reason for measuring the distance between tokens instead of measuring the distance of characters between event types is it reduces the computational overhead that is normally required. The approach is compared with the normal filtering technique on three different log-files collected on two different cluster systems. The results show that the iterative clustering technique achieved higher precision rates and recall rates over the normal filtering technique.

2.3 Feature Selection

In this section, we present the tools that implemented feature selection techniques to process the textual system event logs. Feature selection involves choosing a subset of features to reduce the dimensionality of the data [63]. There are four key steps in feature selection [38]. The steps are: (i) subset generation, (ii) subset evaluation, (iii) stopping criterion and (iv) results validation. Here, the tools we survey are: (i)

BlueGene/L failure analysis and prediction models [45], (ii) Structure-of-Influence (SIG) graphs [51], (iii) rank correlation for processor failure prediction [58], (iv) LogMaster [21], (v) PREdictor [20] and (vi) 3D root-cause analysis [72].

2.3.1 BlueGene/L failure analysis and prediction models

The work reported in reference [45] study RAS (Reliability, Serviceability and Availability) event logs from an IBM BlueGene/L cluster system. The study investigates the characteristics of fatal failure events and the correlation between fatal events and non-fatal events in the event logs. Each record in the BlueGene/L event logs has eight attributes. The attributes are RECID, EVENT_TYPE, FACILITY, SEVERITY, EVENT_TIME, JOB_ID, LOCATION and ENTRY_DATA. The attribute FACILITY denotes the component (e.g., HARDWARE, KERNEL, APP) where the event is flagged. The attribute SEVERITY denotes the level of severity of the event. The levels of severity include INFO, WARNING, SEVERE, ERROR, FATAL or FAILURE. The attributes RECID denotes the record ID, EVENT_TYPE denotes the mechanism through which the event is logged, EVENT_TIME contains the timestamp of the event, JOB_ID denotes the job that detected the event, LOCATION denotes where an error took place, and ENTRY_DATA contains a description of the event.

The event logs contain a lot of records and many of the records are repeated and redundant. To isolate the unique failures, the authors developed a filtering tool that has three steps. In the first step, records that contain the severity levels FATAL or FAILURE are extracted. Using information from the *ENTRY_DATA* attribute, the failures are classified into five types. The types are memory failures, network failures, application I/O failures, midplane switch failures, and node card failures. In the second step, failures that occur within the same sub-system and reported at the same location and by the same job are grouped together in a cluster if the time-window between the failure events is within five minutes; this step is called temporal compression at a single location. In the third step, failures that occur below a time-window of five minutes and reported by the same job but at multiple different locations are removed; this step is called spatial compression across multiple locations.

The authors developed three prediction algorithms. The first prediction algorithm is based on the time-between-failures. The second prediction algorithm is based on spatial skewness. The third prediction algorithm is based on the occurrence of non-fatal events. The results show that the prediction algorithms predicted around 80% of network and memory failures, and 47% of the application I/O failures.

2.3.2 Structure-of-influence graphs

A. Oliner et. al. [51] propose a method for identifying the sources of problems in complex production systems. The method infers the influences among components in a system by looking for pairs of components with time-correlated anomalous behaviour. The influences are summarised in a structure called Structure-of-Influence Graphs or SIGs. A pair of components in a system is defined as two components share an influence when their anomaly signals are correlated. The correlation arises from interactions between the components and the interactions may include direct communication or resource contention between the component s. However, not all the interactions are instantaneous. Hence, effect delays are used to establish the direction of influence. An effect delay is defined as how long an anomaly in one component takes to manifest itself in another component. A SIG encodes the influence as an edge between components and uses the effect delay as a direction of influence.

A Structure-of-Influence graph is constructed using a process that has four steps. In the first step, two models are chosen for producing SIGs. The two models are timing model and entropy model. The timing model is based on message timing where timing behaviour and the classes of events are important. The timing model keeps track of past inter arrival time. The entropy model is based on the information content of message terms where logging is highly selective and ad-hoc. The entropy model keeps track of the distribution of message contents. In the second step, the behaviour of components in terms of surprise, i.e., anomaly signal, is computed. To compute an anomaly signal, the histogram of a recent window of behaviour to the entire history of behaviours for a component is compared. The Kullback-Leibler divergence produces a weighted average of how much the fraction of measurements in a recent distribution differs from the historical distribution. The anomaly signal is produced using the Kullback-Leibler divergence. In the third step, the Pearson correlation coefficient of the anomaly signals of two components with a delay of t time-steps is computed. Then, two matrices are constructed. One matrix is the correlation matrix that contains the correlation coefficients of pairs of components. The second matrix is the delay matrix that contains the offsets of pairs of components. In the fourth step, a SIG is constructed. A SIG is defined as a graph that contains one vertex per component and edges that represent influences. The edges in a SIG indicates the delay associated with the influences, and the delay may be undirected, directed or bi-directional. The correlation matrix and delay matrix are used to construct a SIG.

The authors evaluated the SIGs by using simulation experiments on idealised systems that consist of linear chains of components. The simulations use three types of components. The components are sources, tasks and resources. The component

sources generates data and the component *tasks* processes data. The components sources and tasks require *resources* to generate data and process data. Results from the simulation experiments show that SIGs are robust against uniform message loss, it degrades gracefully when timing measurements are noisy, and it does not depend on clean training data to detect influences.

The authors applied SIGs to diagnose problems in the STANLEY and JUNIOR autonomous vehicles and the Thunderbird supercomputer. A component shared by the lasers on the autonomous vehicle was identified as a cause of the vehicle swerving and a bug was traced to a buffer shared by the lasers. On the Thunderbird supercomputer, the SIG identified a bug in the Linux kernel that caused it to skip interrupts under heavy network activity. The bug caused the kernel to believe that the clock frequency had changed which led the kernel to generate a CPU error. The results showed that the error shared an influence with other nodes in the same job scheduling group.

2.3.3 Rank correlation for processor failure prediction

F. Salfner et. al. [58] propose an architecture for monitoring cores across a chip for dependable process management in chip-multiprocessing machines. A trend in the design of commodity processors is combining multiple execution units on one chip. It becomes more likely that a single execution unit on a processor can fail. To anticipate hardware failures, the processor performance counters are analysed using a statistical rank-sum test. The cores on a chip monitor each other and predict upcoming failures by analysing hardware event sampling data.

First, correlations among the counters available are analysed and the counters that exhibit similar behaviour to other counters are removed. The Spearman-Rank correlation is used because it has the advantage of not assuming any frequency distribution of the variables. Events observed as monotonically increasing would be exhibiting similar behaviour and removed. The result is a list of unique event-type pairs that can be monitored during run-time. Second, two sampling approaches are considered. The sampling approaches are time-based sampling and workload-based sampling. Modern processors do not have a fixed timing behaviour because of frequency scaling, pipelining, and out-of-order execution, and the Pentium time-stamp counter does not guarantee a constant rate for some processor models. On the other hand, the workload-based sampling approach gives a constant scale with respect to the execution of the load application. As such, the workload-based sampling approach was chosen. Third, the Wilcox-Rank sum test is used as the failure prediction approach. It compares a test data set to a reference data set to determine whether the median is about the same. The reference data set contains

CPU counter values that were measured and stored during normal operation without failures. The test data set contains samples that were measured during run-time. The one-sided version of the test is applied. Since no assumption is made about the form of the distribution of the counter values, the one-sided test is considered appropriate for the environment.

The authors performed the experiments with an Intel Core2 Quad CPU (Q6600) with 2.40GHz, 2GB memory and a Linux 2.6 64-bit operating system. A fault injection technique and workload generation is applied to trigger erroneous hardware behaviour. The fault injection technique involved setting the CPU core voltage to a level below normal operation. Overclocking the processor can cause permanent damage to the hardware so it was not used. The workload was generated on the monitored cores using the Mersenne prime number test application MPRIME. 31 performance counters with unique behaviour were investigated and 30 failures for each counter were collected. The area-under-the-curve or AUC metric is used to evaluate the predictors. A perfect predictor would achieve an AUC value of 1 and a random predictor would achieve an AUC value of 0.5. The experimental work showed that there are three groups of event-types. The first group consists of 24 out of 31 counters - this group of counters behaved like purely random predictors. The second and third groups of counters performed better than the first group of counters. In the second group of counters, 5 out of the 31 counters belonged to this group. The AUC values for the counters in the second group are: $AUC_{min} = 0.23, AUC_{median} = 0.56, AUC_{max} = 0.84$. While the counters show high variability in their AUC scores, by inverting the output of the predictor with AUC = 0.23, good predictions could be obtained. 2 out of the 31 counters belonged in the third group. In this group, the counters showed better AUC scores. The scores are: $AUC_{min} = 0.68, AUC_{median} = 0.79, AUC_{max} = 0.91$. While the counters in the third group showed better AUC scores, the variability is still quite high. The authors posed an open question as to whether or not these counters can be turned into good predictors.

2.3.4 LogMaster

LogMaster [21] is a set of innovative algorithms that mines event correlations that have multiple attributes for fast failure prediction. Examples of the attributes include node-id, application-id, event type and event severity. LogMaster proposes two algorithms. The first algorithm, named *Apriori-LIS* and its improved version *Apriori-simiLIS* mine event rules. The event rules are represented by the second algorithm called *Events correlation graphs*. LogMaster was validated on the system event logs on an IBM BlueGene/L system, a 260-node Hadoop cluster system and a

HPC cluster system.

The LogMaster architecture has three major components. The components are Log agent, Log server and Log database. The Log agent collects, preprocess and filters repeated events and periodic events on each node. The Log server mines event rules from events sent by the Log agent to the Log server. The Log server also constructs the event correlation graphs which are used to predict events or failures. A Log database stores the event rules.

A 3-step approach was implemented to mine the event correlations. In the first step, the logs are preprocessed and filtered. Logs in different formats are parsed into a sequence of events where each event is identified by a nine-tuple. Repeated events are identified when the events are recorded by different sub-systems and when the events occur repeatedly in a short time window. Periodic events are identified when the events occur with a fixed interval. Then, repeated events and periodic events are removed. In the second step, the *Apriori-LIS* and *Apriori-simiLIS* event correlation mining algorithms are proposed. The Apriori algorithm uses the apriori property which states that any subset of frequent itemset must be frequent. However, log entries that are represented by event sequences with timing orders are different. Mining for event correlations between log entries with the apriori algorithm can increase the analysis time. To improve the efficiency of the apriori algorithm, an event filtering policy is applied before mining the event correlations. The event filtering policy reduces the number of events that has to be analysed by extracting events that occur in the same nodes, the same application or have the same event types. In the third step, *event correlation graphs* for representing event rules is proposed. An event correlation graph is a directed acyclic graph where a vertex represents an event and an edge represents the correlation of two events linked by the edge. Vertices can be dominant or recessive. For a $2-ary$ event rule, the vertices A and B that represent two events are dominant. In the case when events A and B occurred and event B occurred after event A, the vertex $A \wedge B$ is a recessive vertex. Edges link vertices and edges can be dominant or recessive. An edge is dominant if the two vertices that are linked by the edge are dominant. Otherwise, the edge is recessive. Three main steps are used in constructing an event correlation graph. In the first step, a group of event correlation graphs based on event rules found during single-node analysis is constructed. In the second step, event correlation graphs that represent event correlations found during multiple-node analysis are constructed. In the third step, indices that record the positions of events in the event correlation graphs are saved. The indices are used to locate the events in the event correlation graphs.

The authors applied LogMaster to analyse three real logs generated by an IBM

BlueGene/L cluster system, a Hadoop cluster system and a HPC cluster system. To evaluate the algorithms, three evaluation metrics are defined. The first metric is the average analysis time, the second metric is the average prediction time, and the third metric is the precision and recall. The results from event preprocessing and filtering show that a compression rate of above 90% is achieved on the Hadoop and IBM BlueGene/L logs, and a compression rate of 69.4% is achieved on the HPC cluster system logs. The results from event rules mining show that the *Apriori-simiLIS* algorithm significantly improves the time efficiency of the *Apriori-LIS* algorithm with small rule loss. The results from prediction show that the precision rates of the *Apriori-simiLIS* algorithm is higher than the *Apriori-LIS* algorithm, and the recall rates of the *Apriori-simiLIS* algorithm is lower than the *Apriori-LIS* algorithm. The reason for the high precision rates and low recall rates is due to keeping a richer set of log information without spatial filtering. The results from average prediction time of the three logs show that the *Apriori-simiLIS* algorithm has a shorter prediction time than the *Apriori-LIS* algorithm.

2.3.5 PREdictor

The work reported in reference [20] presents the development of two models for predicting failures in coalition systems. The first model is a spherical covariance model and the second model is a stochastic model. The spherical covariance model has an adjustable timescale parameter and it is used to quantify the temporal correlation among failure events. The distance in time between two failures is used to calculate their covariance value and the timescales for calculating the covariances can be adjusted for different types of failures. The probabilistic distribution of failures is used in the stochastic model to compute the spatial covariance among failures. A framework called PREdictor is implemented to explore correlations among failures and forecast the time-between-failure of future instances.

The failure prediction framework has a multi-layer prediction architecture for analysing the correlations of failure instances in different scopes of a coalition system. The framework has node-wide, cluster-wide and system-wide failure predictors. The node-wide failure predictor keeps track of the new events recorded to the local event logs since its last operation, extracts failure records, creates formatted failure reports, monitors the performance dynamics of executing applications and measures the resource utilisation. The cluster-wide failure predictor collects failure reports from the compute nodes managed by the master node, statistically processes and analyses the failure events, predicts prospective failures and generates system availability reports for the resource scheduler and system administrator. The system-wide failure predictor receives failure reports from master nodes of clusters and forecasts the

failure dynamics of the entire coalition environment for system management.

The performance of the prediction framework in both offline and online prediction modes was evaluated on the Los Alamos HPC traces and in an institute-wide clusters coalition environment. The results show that the prediction framework can achieve an accuracy of more than 76% in offline prediction and more than 70% accuracy in online prediction.

2.3.6 3D root-cause analysis

Z. Zheng et. al. [72] present an automated root-cause diagnosis mechanism for large-scale HPC systems. The diagnosis mechanism provides fine grained root-cause analysis by pinpointing the failure layer, the time and location of the event that caused the failure. The authors evaluate their diagnosis mechanism on real RAS (Reliability, Serviceability, Availability) logs collected from a production IBM BlueGene/P system at Oak Ridge National Laboratory.

The diagnosis mechanism is comprised of four interrelated steps. The steps are: (i) preprocessing, (ii) information fusion, (iii) layer identification and (iv) time and location identification. The preprocessing step tackles the challenge of data volume by removing redundant records and noise from the RAS logs. The information fusion step synthesizes the information from RAS, job and environmental logs. The layer identification step reduces the search space by co-analysing multiple logs on the application, system software and hardware layers. The time and location identification layer pinpoints the event which triggered the failure by tracing the event that occurred days before the failure.

The authors presented four case studies identified using the diagnosis mechanism. The case studies are a hardware BPC clock failure, an application out of memory failure, a network torus sender failure and a kernel panic. A node power error was identified as a trigger event for the BPC clock failure. An insufficient memory error was identified as a trigger event for the application out of memory error. An invalid memory address error was identified as a trigger event for the network torus sender failure. A machine check error was identified as a trigger event for the kernel panic.

2.4 Anomaly Detection

In this section, we present anomaly detection tools that implement feature extraction as the main technique for processing system usage logs, textual event logs or both system usage logs and textual event logs. The tools are: (i) automated anomaly detection [44], (ii) adaptive anomaly detector [27], (iii) increasing error handling

time-window [33] and (iv) DILAF [2].

2.4.1 Automated anomaly detection

The work reported in reference [44] present an automated mechanism for node-level anomaly identification in large cluster systems. The mechanism is comprised of a set of techniques that analyse data collected from many sensors. It transforms the data into a form suitable for feature extraction, extracts features from the data and identifies anomalous nodes in an unsupervised manner. Two feature extraction techniques are compared. The techniques are Principal Component Analysis (PCA) and Independent Component Analysis (ICA). The authors implemented a prototype and injected a variety of faults into a production system at the National Center for Supercomputing Applications to evaluate the prototype.

A total of 19 features were collected per node for the experiments. The features that were collected include CPU, memory, I/O and network metrics. To evaluate the PCA and ICA-based anomaly detector, the authors defined two sets of metrics. The first metric, *sensitivity*, measures the proportion of correct faulty classifications to the number of actual faulty nodes. The second metric, *specificity*, measures the proportion of correct non-faulty classifications to the number of actual normal nodes.

The authors conducted two sets of tests. The tests are a single-fault test and multifault test. In the single-fault test, one type of faults is injected into 0 to 20 randomly selected nodes. Then, the PCA-based and ICA-based implementation is assessed as to whether it can correctly identify the faulty nodes. The authors compared results obtained on the PCA, ICA and no feature extraction based methods. Their results show that the feature extraction based methods achieve better sensitivity and specificity than the no-feature extraction based method. Their results also show that the ICA-based method achieves the best sensitivity and specificity. In the multifault test, different types of faults are simultaneously injected into 0 to 20 randomly selected nodes. The pairs of faults include memory and CPU, CPU and network, I/O and network. The results obtained show that the ICA-based method achieves a specificity of above 0.94 and a sensitivity of 1. The ICA-based method outperforms both the PCA-based method and the no-feature extraction based method in terms of specificity and sensitivity.

2.4.2 Adaptive anomaly detector

The work reported in reference [27] present an adaptive anomaly detection mechanism that develops Principal Component Analysis (PCA) to identify anomalous behaviour

in cloud computing systems. The anomaly detection mechanism integrates cloud performance metric analysis with filtering techniques to extract the most relevant principal components of different types of failures. The authors implement a prototype of the anomaly detector and conduct experiments using traces obtained on a Google data centre.

The adaptive anomaly detection mechanism is comprised of two algorithms. The first algorithm, called Most Relevant Principal Component (MRPC), identifies a set of principal components that have strong correlation with failures. It uses neural networks to compute the principal components from normalised values of cloud performance metrics in a rolling time window. The second algorithm, called Adaptive Anomaly Identification, identifies anomalies using MRPCs by leveraging adaptive Kalman filters.

The authors conducted experiments on a cloud testbed that consists of 362 servers. Third party monitoring tools such as sysstat and perf were used to collect runtime performance data on the hypervisor and virtual machines. The authors also develop a fault injection program to inject a variety of faults into the cloud testbed. They studied four types of failures identified by the MRPC algorithms. The failure cases are memory related failures, disk related failures, CPU related failures and network related failures. The MRPC-based anomaly detector is compared with decision tree, Bayesian network, support vector machine (SVM) and PCA-based anomaly detectors. Their results show that the MRPC-based detector achieves the best performance, with a true positive rate of 91.4% and false positive rate of 3.7%.

2.4.3 Increasing the error handling time window

N. Gurumdimma et. al. [33] address a fundamental question, that is, increasing the error handling time window in large-scale distributed systems by using resource usage logs and system failure logs. The resource usage logs are used to track anomalous resource usage and the system message logs are used to identify the root-causes of system failures. An anomaly-based detection algorithm is developed to identify anomalous resource usage in the system.

The methodology has two phases. The first phase identifies events that are correlated with frequently occurring failures. Fault events that are regarded as causes of system failures are extracted from large logs of cluster systems using an existing fault diagnostics tool reported in reference [7]. In the second phase, an approach that extracts anomalous running jobs believed to be pointers to problems in the cluster system and correlated in the event logs is explained. The approach has three steps. In the first step, the counters for each job on each node within a given time are extracted from the resource usage logs. Then, a Resource Usage Feature

Matrix is generated. The resource use counters are represented by the columns of the matrix and the jobs on each node are represented by the rows of the matrix. In the second step, an unsupervised approach based on Principal Component Analysis is introduced. In the approach, the outlierness of a job is determined by the variation in the dominant principal direction. The variation is computed by subtracting the cosine similarity between two leading principal directions by 1. A job and an earliest time when the anomaly occurred are saved when the variation exceeds a specified threshold. The authors showed that PCA can identify anomalous jobs in the resource usage logs but it could not identify relationships among the resource use counters. In the third step, the Maximal Information Coefficient is applied to understand the relationships between the resource use counters. Then, the error event lead time is defined. It records the time when the earliest anomalous job is detected to the time when the fault is logged.

The authors evaluated their approach using the resource usage logs and system message logs collected on the Ranger supercomputer at the Texas Advanced Computing Center. Their results show that the lead time to failure can be extended by up-to 55 minutes.

2.4.4 DILAF

DILAF [2] is a framework for distributed analysis of large scale system event logs for anomaly detection. The framework is comprised of three main processes which facilitate log parsing, feature extraction and machine learning activities. DILAF distinguishes itself from existing tools by not requiring the availability of source codes in the analysed system and performing all the processes in a distributed manner to support scalable analysis.

The architecture of DILAF is comprised of four processes. They are: (i) log parsing, (ii) feature extraction, (iii) normalisation and (iv) machine learning. The log parsing process transforms the free-form log messages into structured and featured events. The feature extraction process constructs numerical feature vectors and creates a primary message count vector utilising the identifier and message type information, which are extracted by the log parsing process. The normalisation process applies Term Frequency/Inverse Document Frequency (TF/IDF) technique that measures the importance of a message type for an identifier and generates a feature matrix. The machine learning process applies Principal Component Analysis that filters repeating patterns in the feature matrix to identify anomalous patterns. The authors implement PCA using the Bulk Synchronous Parallel computation model to distribute the computational workload among participating processes. To ensure the accuracy of the anomaly detector, they use Apache's Spark Resilient

Distributed Datasets to recover a partition in the event of a crash.

The authors conducted experiments on a Hadoop Distributed File System (HDFS) log dataset and the Thunderbird supercomputer logs. On the HDFS dataset, the DILAF framework achieved 99.8% accuracy for anomaly detection. On the HDFS dataset, DILAF achieved a performance increase of 65% over the log parsing method reported in reference [69]. On the Thunderbird supercomputer dataset, DILAF achieved a performance that scales linearly with respect to the size of the logs; the Thunderbird supercomputer dataset is 20 times larger than the HDFS dataset.

2.5 Hybrid Methods

In this section, we present the tools that develop hybrid techniques for processing the textual system event logs. The tools are: (i) Error log processing [57],(ii) System log pre-processing [70], (iii) LogSig [47], (iv) LKE [19] and (v) LogAider [16]. A hybrid technique uses two or more methods from regular expressions, clustering, correlation and feature extraction.

2.5.1 Error log processing for accurate failure prediction

F. Salfner and S. Tschirpke [57] propose three algorithms that show that data preparation is an important step to achieve accurate error-based online failure prediction. The first algorithm assigns error IDs to error messages by using Levenshteins edit distance. The second algorithm groups similar error sequences by using a sequence clustering technique. The third algorithm filters statistical noise by using a filtering algorithm. The algorithms were evaluated on error logs derived from a commercial telecommunications system. Their results show that the failure prediction accuracy drops by up-to 45% when the original logs are used.

The first algorithm automatically assigns error IDs to error messages using Levenshteins edit distance. The error ID captures the type of error message. For example, in the following error log `process 1534: end of buffer reached` the number 1534 relates to the source rather than the type of message. All numbers and log-record specific data such as IP-addresses are replaced by placeholders. A copy of the original error log is kept to maintain the information. However, a 100% replacement of all record-specific data is infeasible because there are many typographical errors. To address this issue, the Levenshteins edit distance is computed between all pairs of error messages that appear in the logs. The method of substituting numbers by placeholders produced a reduction in the number of original messages by 99.26%. The Levenshteins edit distance method produced a reduction in the number of original messages by 99.92%.

The second algorithm uses a method called Tupling [64] to group similar error sequences. The Tupling method groups error events that occur within a time interval (temporal tupling) or error events that refer to the same location (spatial tupling). For the telecommunications system being studied, spatial tupling is not considered because it works only for systems with strong fault containment regions. The tupling method groups all errors that show an interarrival time less than a predefined threshold. The authors use three parameters to extract the sequences. The parameters are lead time, data window size and margins for non-failure sequences. The first parameter, lead time, extracts failure sequences that precede the failure occurrence by a time of 5 minutes. The second parameter, data window size, determines the length of each sequence by a maximum time of 5 minutes. The third parameter, margins for non-failure sequences, extracts non-failure sequences when the system is fault free and applies a ban period of 20 minutes before and after a failure. The failure sequence tuples are used to train a hidden semi-markov model and obtain a dissimilarity matrix. Then, hierarchical clustering methods are applied to the dissimilarity matrix and obtain groups of similar sequences. The actual number of groups is determined by visually inspecting the banner plots.

The third algorithm removes unrelated events in the groups of similar sequences produced by the second algorithm. Unrelated events in the group of sequences occur mainly due to system processes which execute concurrently to produce the log messages. The filtering method is implemented using a test of goodness of fit. It is based on the notion that indicative events occur more frequently within a failure sequence by the same failure mechanism than within other failure sequences. In failure sequences of the same cluster, each error is checked for significant deviation from the prior using a test variable defined as the non-squared standardised difference. When the test is less than a predefined threshold, the error is eliminated from the failure sequence.

2.5.2 System log pre-processing for improving failure prediction

Z. Zheng et. al. [70] present a system log pre-processing method for preserving important failure patterns that are crucial for failure analysis. The pre-processing method consists of three integrated steps. The steps are: (i) event categorisation, (ii) event filtering and (iii) causality-related filtering. The effectiveness of the pre-processing method is demonstrated on real failure logs collected on a Cray XT4 system and an IBM BlueGene/L cluster.

The pre-processing method addresses three issues of system log pre-processing. The issues are: (i) finding important failure patterns, (ii) identifying events with multiple spatial locations, and (iii) characterising different aspects of a failure. The

first step, event categorisation, uses regular expressions to classify various events into a hierarchical set of event categories. The second step, event filtering, uses an improved temporal and spatial filtering method that keeps track of event start and the end times, event count and event location. The third step, causality-related filtering, adopts apriori association rules to track causal correlations among events. The authors conducted experiments on failure logs collected on a Cray XT4 system and an IBM BlueGene/L cluster. The results show that their pre-processing method effectively preserves failure patterns and improves failure prediction by up-to 174%, with a compression rate of more than 90%.

2.5.3 LogSig

LogSig [47] is a message based signature algorithm that generates system events from raw textual message logs. It categorises log messages into a set of event types by searching for the most representative message signatures, and it is able to incorporate human domain knowledge to achieve a high performance. The goal of LogSig is to identify the event type of each log message according to a set of message signatures. To achieve this, a metric called *Match Score* is proposed to determine which signature best matches a log message. The metric computes the Multiple Longest Common Subspace (MLCS) between multiple sequences. However, the Multiple Longest Common Subspace is known to be a NP-hard problem. The authors proved by contradiction that the MLCS problem can be reduced to the Longest Common Subspace problem between two sequences, and proved by contradiction that the Longest Common Subspace problem can be solved in polynomial time with a polynomial time solution.

The authors present an approximated version of MLCS and the LogSig algorithm. The LogSig algorithm is comprised of three steps. In the first step, every log message is separated into several pairs of terms. In the second step, a search strategy is used to find groups of log messages where each group share many common pairs. In the third step, message signatures based on identified common pairs in each message group is constructed. The authors compared LogSig with seven algorithms on five different system logs. The algorithms are IPLoM, VectorModel, Jaccard, StringKernel, StringMatch, semi-StringKernel and semi-Jaccard. LogSig outperformed the seven algorithms in terms of overall performance.

2.5.4 LKE

The work reported in reference [19] propose an unstructured log-analysis technique for anomaly detection. In the technique, a novel algorithm that converts free form

text messages in log-files to log-keys is proposed. The algorithm does not rely heavily on knowledge of the application-domain. After the messages have been converted to log-keys, a Finite State Automaton (FSA) is learnt to present the normal work flow for each system component. A performance measurement model is then learnt to characterise normal execution performance based on the timing information in the log messages. The authors applied their technique on the Hadoop and SILK distributed computing systems and showed that their algorithm can detect running anomalies effectively.

The technique consists of two processes: (i) the learning process, and (ii) the detection process. The learning process obtains models that represent the normal execution behaviour of the system from logs produced by normally completed jobs. The learning process consists of three steps: (1) log message sequences are converted into log key sequences, (2) a Finite State Automaton is derived to model the execution path of the system, and (3) the execution time of each state transition is counted and a performance measurement model is obtained through statistical analysis. The reason to convert log message sequences into log key sequences is to address the problem of mining high dimensional data when directly considering log messages as a whole. The log key is defined as the common content of all log messages which are printed by the same log-print statement in the source code; in other words, a log key is equal to the free form text string of the log-print statement minus any parameters. A clustering approach based on measuring the similarity of two raw log keys is implemented. The approach clusters the raw log keys into initial groups, followed by splitting the groups. The position of words in the raw log keys are used to measure the similarity of two raw log keys because most programmers tend to write text messages first then the parameters are added later. The raw log keys similarity is measured by the weighted edit distance in which the sigmoid similar function is used to compute weights at different positions. A threshold is determined to automatically connect pairs of raw log keys together. The k-means algorithm is used to cluster all distances into two groups. The distances correspond to the inter-class and inner-class distances. The largest distance from the inner-class distance group is selected as the threshold. An existing algorithm is applied to learn a FSA from sequential log sequences. Each transition in the learned FSA corresponds to a log key and a state is represented by a log message.

A performance measurement model is derived to characterise the performance of the normally completed jobs. Each log key sequence is first converted to its corresponding state sequence. Then the state time-stamp is specified by the time-stamp of its corresponding log key in the log key sequence. Two performance models are defined to measure low performance problems. The first model called *transition*

time measurement model measures the time interval that a system component takes to transit from one state to the next state which takes much longer than normal cases. The second model called *circulation numbers measurement model* counts the number of circulations in a loop structure. The transition time between adjacent states and the circulation numbers of all loop structures are used to characterise the normal performance of jobs.

2.5.5 LogAider

S. Di et. al. [16] develop a tool called LogAider for mining event correlations in HPC system event logs. LogAider reveals three types of potential correlations. The correlations are: across-field, spatial and temporal correlation. LogAider's design architecture is comprised of four layers. The layers are: user interface layer, analysis engine layer, log parsing layer and log data layer.

The log data layer includes system RAS and job logs. The system logs are stored in a relational database. The log parsing layer performs preliminary processing of the log data, which includes reading the schema information to recognise the meaning of each field and filtering out duplicate messages. The analysis engine layer performs across-field, spatial and temporal correlation. The across-field correlation engine computes: (i) the prior probability distribution based on various metrics or fields in the logs, and (ii) the posterior probability based on any combination of fields set by the user. The spatial correlation engine mines potential correlations across multiple components using an optimised K-means algorithm. The temporal correlation engine mines potential correlations using similarity analysis of events that occur close in time. The user interface layer provides templates for specifying the key fields for performing across-field, spatial and temporal correlation analysis to the user.

The authors performed experiments on one year worth of RAS and job logs obtained on the Mira supercomputer at Argonne Leadership Computing Facility. Their results show that: (i) across-field correlation analysis achieved precision and recall rates of 99.9%, and (ii) temporal correlation analysis achieved an accuracy of 95%.

2.6 Similarities and Differences

In this section, we summarise the similarities and differences between the system log-file processing tools surveyed. A comparison table for the system log-file processing tools is given in Table 2.1.

Table 2.1: Comparison table for system log-file processing tools.

Tool reference	**Resource use data**	**System logs**	**Feature selection**	**Feature extraction**
[54]	No	Yes	No	No
[56, 65]	No	Yes	Yes	No
[37]	No	Yes	No	No
[53]	No	Yes	Yes	No
[66]	No	Yes	No	No
[48]	No	Yes	No	No
[23]	No	Yes	No	No
[49]	No	Yes	No	No
[52]	No	Yes	Yes	Yes
[32]	No	Yes	No	No
[45]	No	Yes	Yes	No
[51]	No	Yes	Yes	No
[58]	Yes	No	Yes	No
[21]	No	Yes	Yes	No
[20]	No	Yes	Yes	No
[72]	Yes	Yes	Yes	No
[33]	Yes	Yes	No	Yes
[44]	Yes	No	No	Yes
[27]	Yes	No	No	Yes
[2]	No	Yes	No	Yes
[57]	No	Yes	No	No
[70]	No	Yes	No	No
[47]	No	Yes	No	No
[19]	No	Yes	No	No
[16]	No	Yes	Yes	No

The similarities between the system log-file processing tools reviewed are given as follows:

- All the tools develop pre-processing steps which can be summarised into two activities. The first activity tokenises the system log messages by extracting sequences of words in the free form text in the system log message. The second activity applies some method to measure the similarity between the system log messages.

- Each tool implements pre-processing that comprises multiple steps. In each pre-processing step, the messages in the system log-files are grouped into initial groups. Then, the messages in each of the initial groups are further grouped into smaller groups.

- All the tools extract sequences of English-only words in the free form text

in the system message logs and discard tokens that contain only numbers or alpha-numeric words.

The differences between the system log-file processing tools reviewed are given as follows:

- Each tool develop a different technique. The techniques can be divided into five groups. The groups are: (i) regular expressions, (ii) clustering, (iii) feature selection, (iv) anomaly detection, and (v) hybrid method. A hybrid method combines two or more techniques.

- Majority of the tools surveyed target general system event log-processing. By general, we mean that the tool does not specifically focus on a dependability-centric issue such as failure prediction.

- Differently to the generic system log-file processing tools, the tools reported in references [14, 15, 20, 21, 45, 52, 57, 58, 70] develop methods that target failure prediction. The work reported in references [2, 27, 44] have developed system logs or resource usage data processing methods that target detection of faulty nodes. N. Gurumdimma et. al. [33] developed a methodology for increasing the error handling time-window. A. Oliner et. al. [51] and Z. Zheng et. al. [72] developed different methods for diagnosing cluster system failures. S. Di et. al. [16] developed LogAider that targets propagation of errors that lead to system failures.

In Chapters 4, 5 and 6, we present two new system diagnosis workflows. The workflows are:

- We developed the CORRMEXT and EXERMEST frameworks for HPC systems diagnosis.

- CORRMEXT integrates data type extraction, correlation and time-bin extraction methods. It identifies frequently occurring error cases.

- EXERMEST integrates multiple feature extraction and correlation methods. It identifies rare error cases.

2.7 Summary

We presented a detailed survey of system log-file processing tools. We identified three similarities between the tools and showed that the pre-processing steps implemented in all the tools can be summarised according to two activities. The first activity

tokenised system log messages by extracting sequences of English-only words in the free form text of the textual message logs. The second activity applied a method to measure the similarity between system log messages. We highlighted the similarities and differences between the system log-file processing tools.

In the next chapter, we will describe the system and fault models, explain the system issues, give an overview of the Ranger, Lonestar4 and Stampede-1 HPC systems and the log-data on these HPC systems, and describe the implementation details for the cluster log-data preprocessing modules.

Chapter 3

System Models, HPC Systems and Cluster Log-Data

Modern day data centres and high performance computing (HPC) systems are made up of complex combinations of processors, networks, operating system processes, memory and storage systems. When new technologies are introduced, the behaviour of these computing systems can change rapidly. The computing systems generate a lot of data and different types of data. For the system administrators, manually scanning the system logs to identify the cause of a system failure is a labour intensive task. In this chapter, we present the problem and describe the HPC systems and cluster log-data.

We structure this chapter as follows: In Section 3.1, we describe the system model for which the CORRMEXT and EXERMEST frameworks can be applied. In Section 3.2, we introduce the Ranger, Lonestar4 and Stampede-1 HPC systems operated by the Texas Advanced Computing Center. In Section 3.3, we describe the TACC_Stats resource use data [17], Rationalized message logs [36] and Syslogs [40]. In Section 3.4, we give the implementation details for the data preprocessing module.

3.1 System Model

A general HPC systems model is specified in [8] and we describe the system model here. The CORRMEXT and EXERMEST frameworks can be applied to a generic HPC system model as follows: A HPC system S is comprised of X jobs $J_1...J_X$, Y nodes $N_1...N_Y$ and Z production time-bins $T_1...T_Z$. A HPC system uses a job scheduler JS for: (i) allocating jobs to nodes and communicating paths between nodes, and (ii) setting job production times. The HPC system executes a heterogeneous system software stack that is comprised of a filesystem FS, operating system OS, network

software NS and operating system processes $OS-PS$. We assume that all the nodes in the HPC system synchronise their clocks and each node maintains its own clock. System logs may be written to containers $U_1...U_n$ by each job, node, job scheduler and system software. System resources that are used by the nodes, jobs and system software may be written to containers $W_1...W_m$ as resource use logs. Data may be transferred to and from the nodes, jobs, system software and filesystem.

The privacy policies of a data centre may not allow access to software codes. Therefore, we assume that there is no access to software codes in a data centre. However, a data centre may allow access to system logs and resource use data. The system logs contain system and failure events. The resource use data contains system resource utilisation counters. For the objective of diagnosing failures, i.e., to determine where, when and why the system crashed, the data centre may grant access to the system logs, resource use data and system administrators for validating the diagnoses. However, a data centre may not grant access to the system maintenance records due to security reasons. Therefore, we assume that the operation context of the HPC system is not available. Our HPC system model covers both open source Linux-based computational facilities as well as bespoke computing systems, for example IBM BlueGene systems.

3.1.1 Fault Model

When the system output deviates from the expected one, a system failure occurs. We define a system failure as a node crash or operating system hang-up. In order to increase the dependability of the system, it is important to tolerate those errors that exist shortly before a system failure. Tolerating errors mean to detect them and then to correct them, if possible. However, typically before a system failure, an error has propagated beyond the interface of a source component to affect multiple components. Thus, knowledge about these errors can be useful to system designers and implementors for improving the effectiveness of error recovery protocols.

We assume faults can occur at any level in the system, at the lowest level, for example, register level to the highest level such as operating systems. When these faults are executed, it will lead to errors which may lead to a system failure if the error is not handled. Our objective is to identify error cases without prior knowledge of the fault models.

3.1.2 System Issue

Figure 3.1 shows an illustration of resource use counters and error messages by time. The resource use counter `rx_bytes` records the amount of network data received.

The resource use counter `rx_crc_errors` records the amount of CRC errors received. The resource use counter `rx_frame_errors` records the amount of frame errors received. In the resource use data, we observe that `rx_bytes` and `rx_crc_errors` are correlated. Furthermore, we observe that `rx_bytes` and `rx_frame_errors` are correlated. The message log `master network unreachable` indicates a DNS lookup failure. The message log `FTP failed` indicates a file transfer protocol failure. In the system logs, we observe that `master network unreachable` and `FTP failed` are correlated.

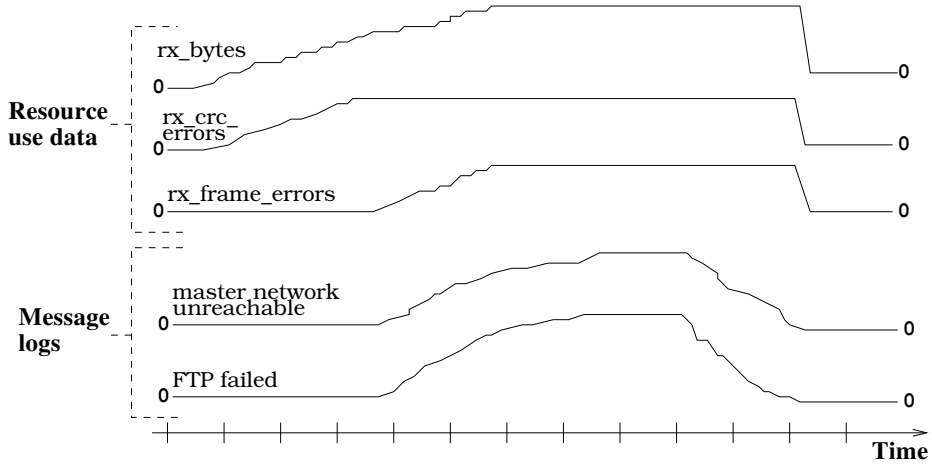


Figure 3.1: An illustration of rx_bytes, rx_crc_errors and rx_frame_errors resource use counters, master network unreachable and FTP failed messages.

Capturing these correlations is important as these can provide insights into the behaviour of the system. From Figure 3.1, the resource use counters r_1 and r_2 may capture the existence of two errors in the system while another counter r_3 may capture the normal behaviour of the system. A correlation between r_1 and r_3 and a correlation between r_2 and r_3 shows that the two errors were triggered following a normal system behaviour. Similarly, a correlation between different event groups will provide insights into the system state. However, understanding the occurrence of groups of errors from system logs alone is challenging. In this thesis, we capture the notion of an error when: (i) a message is logged and the message captures a state of the program that deviates from expectation, and (ii) there are groups of resource use and error message patterns as illustrated in Figure 3.1. In Chapters 4 and 5, we will present the CORRMEXT framework that seeks to determine the occurrence of these patterns to study errors which lead to a system failure or recovery.

3.1.3 Significant Errors on Nodes

Let S be a HPC system with n distinct nodes and all the nodes in the system S are linked together by a network. We capture the significant errors by first defining a set $NS_i = \{n_1, n_2, ..., n_i | n_i \in Node_{state}\}$, where $1 \leq i < x$, x is the number of nodes, $Node_{state} = \{n_{fault}, n_{error}, n_{failed}\}$, n_{fault} = a node on which a fault is triggered, n_{error} = a node that contained an error, n_{failed} = a node that crashed. Thus, an error can occur within one node or occur on two or more nodes.

Due to the possibility that errors are associated with a large number of nodes, finding the significant errors is challenging. In Chapter 6, we will present the EXERMEST framework that investigates the use of feature extraction methods to identify the significant errors and the nodes associated with the identified errors.

3.2 Case Study HPC Systems

In this section, we present the Ranger, Lonestar4 and Stampede-1 HPC systems operated by the Texas Advanced Computing Center at The University of Texas at Austin.

3.2.1 Ranger

The Ranger HPC system[1] was a Linux-based cluster that consisted of 4,048 nodes featuring AMD quad-core Opteron processors. It was operated from 2007 to 2013. All the nodes were linked together via a high-speed Infiniband network. Job scheduling and resource management were provided by the Sun Grid Engine[2]. High speed file access was provided by the Lustre filesystem[3].

Each node of Ranger generated its own resource use data and rationalised messages. Ranger was the first HPC system operated at a United States academic institution that deployed TACC_Stats [35] and Rationalized message logs [36]. The TACC_Stats resource usage monitor is a job-oriented and logically structured version of the conventional Sysstat system performance monitor. The Rationalized message logs incorporates a logical structure and additional content such as job-identification (rationalisation) to the POSIX formatted logs. The resource usage and rationalized messages generated on the Ranger HPC system nodes were sent to a centralised logging system. The resource use logs were combined and interleaved in time. The Rationalized message logs were combined and interleaved in time. Ranger used UDP (User Datagram Protocol) as its primary communications protocol for all the

[1] `https://www.tacc.utexas.edu/-/ranger-supercomputer-begins-new-life`
[2] `http://web.njit.edu/alltopics/HPC/sge.html`
[3] `http://lustre.org/`

processes that ran on its nodes. The reason for using UDP are lower bandwidth overhead and latency.

3.2.2 Lonestar4

The Lonestar4 HPC system[4] was a Linux-based cluster that consisted of 1,888 compute nodes, with two Intel 6-Core processors per node, for a total of 22,656 cores. It was configured with 44TB of total memory and 276TB of local disk space. Lonestar4 was operated from 2009 to 2015. All the nodes were interconnected with Infiniband technology in a fat-tree topology with a 40Gbit/sec point-to-point bandwidth. All Lonestar4 nodes ran Linux Centos 5.5 and supported batch services through the Sun Grid Engine. Global, data intensive I/O was supported by a Lustre filesystem, while home directories were serviced by an NSF filesystem with global access.

Each node of Lonestar4 generated its own TACC_Stats resource use data and standard Linux syslogs. The resource usage and Linux Syslogs were sent to a centralised logging system. The resource use logs were combined and interleaved in time. The Linux Syslogs were combined and interleaved in time. Lonestar4 used UDP (User Datagram Protocol) as its primary communications protocol for all the processes that ran on its nodes.

3.2.3 Stampede-1

The Stampede-1 HPC system[5] was a Linux-based cluster that consisted of 6,400 nodes featuring Intel Xeon E5 Sandy Bridge host processors and the Intel Knights Corner (KNC) co-processor. It was configured with 260TB of total memory, 14PB of shared disk space, and 1.6PB of local disk space. Stampede-1 was operated from 2012 to April 2018. In the Sandy Bridge cluster, all the nodes were interconnected with Infiniband technology in a fat-tree topology of eight core-switches and over 320 leaf switches. All nodes in the Sandy Bridge cluster ran Linux CentOS 6.3 and were managed with batch services through the Slurm workload manager[6]. Global HOME, WORK and SCRATCH storage areas were supported by three Lustre parallel distributed filesystems with 76 I/O servers. In the KNC cluster, all the nodes were interconnected with a separate Intel OmniPath network. All nodes in the KNC cluster ran Linux CentOS 7.

Each node of Stampede-1 generated its own TACC_Stats resource use data. The resource usage were sent to a centralised logging system. The resource use logs

[4] `https://portal.tacc.utexas.edu/archives/lonestar4`
[5] `https://www.tacc.utexas.edu/systems/stampede`
[6] `https://slurm.schedmd.com/`

were combined and interleaved in time. Stampede-1 used UDP (User Datagram Protocol) as its primary communications protocol for all the processes that ran on its nodes.

3.3 Cluster Log-Data

In this section, we describe the TACC_Stats resource use data [17], Rationalized message logs [36] and Syslogs [40].

3.3.1 Rationalized Message Logs

The Rationalized message log [36] incorporates additional content such as job-identification (rationalisation) and a logical structure to the POSIX formatted logs. Rationalising log messages helps improve the effectiveness of log-based failure analysis of open-source HPC systems. The Rationalized message logs provide: (1) easy comprehension of unstructured log messages, (2) simple parsing and (3) direct mapping of errors and failures to jobs. The structure of a rationalized log message is shown below:

```
time:1273001236
host:i175-110
jobid:1366122
prog:kernel
0:<3>spurious soft lockup detection on CPU#\%d
1:17
...
```

The header of a rationalized log message has four main fields. They are: (i) `time`, (ii) `host`, (iii) `jobid` and (iv) `prog`. The `time` field contains a value that tells the total number of seconds that have elapsed since 1 January 1972 00:00:10. In this example the value is `1273001236`. The `host` field contains the node. In this example the node is `i175-110`. The `jobid` field contains each executed job through an assigned numerical number. In this example the job number is `1366122`. The `prog` field contains the protocol name. In this example the name of the protocol is `kernel`. The `0:` field contains the key event that occurred at the recorded time and on the recorded node and job. In this example the key event is a soft lockup of a central processing unit. The remaining fields provide additional information associated with the key event.

3.3.2 Syslogs

Syslog [40] is a general system and program messages logging system in the Linux environment. The Syslog service is comprised of the system log daemon, where Linux and its programs can send kernel and program messages to. The logs handled by Syslog is available in the `/var/log/` directory on a Linux system. Among the logs in the `/var/log/` directory, the most common one is `/var/log/messages` that stores the kernel system message as well as kernel module core dumps. Thus, the `/var/log/messages` is the main log-file to examine for problem diagnosis and monitoring on Linux-based systems. The POSIX [40] standard for logging system events allows the freedom for formatting logs. As such, the structure of a log message can vary. An example of a Syslog is shown below:

```
Jan 12 15:24:55 oss5 kernel: LustreError: 0:0:(ldlm_lockd.
c:249:waiting_locks_callback()) ### lock callback timer
expired after 254s: evicting client at *.*.*.*@o2ib ....
```

We observe that the above Syslog is comprised of five fields. The fields are: (i) timestamp, (ii) node-id, (iii) system-id, (iv) application-id and (v) error message. Columns one to three contain the date and time. In this example the date and time is `Jan 12 15:24:55`. Column four contains the node-id. In this example the node is `oss5`. Column five contains the system-id. In this example the system is the Linux kernel. Column six contains the application name. In this example the application is the Lustre filesystem. The remaining columns contain the error message.

3.3.3 TACC_Stats Resource Use Data

TACC_Stats [17] is a job-oriented and logically structured version of the conventional Sysstat system resource usage monitor. TACC_Stats provides online monitoring of system resources and it records all the values on all the system metrics it monitors at all time-intervals. TACC_Stats will only reset the system metrics values to zero if an actual reset on a node is performed. An example of a TACC_Stats resource use log is given below:

```
2066522 Aug 11 12:50:01 i150-412 eth0 rx_bytes 302345 ..
```

In a TACC_Stats resource use log, the first column contains the job number. In this example the job number is `2066522`. The second to fourth columns contain the date and time. In this example the date and time is `Aug 11 12:50:01`. The fifth column contains the node-id. In this example the node is `i150-412`. The sixth column contains the component identifier. In this example the component is `eth0`.

The pairs that follow after the component identifier contains the name of the resource use counter and its value. In this example the resource use counter is `rx_bytes` and its value is `302345`. The list of resource use counters is given in Table 3.1.

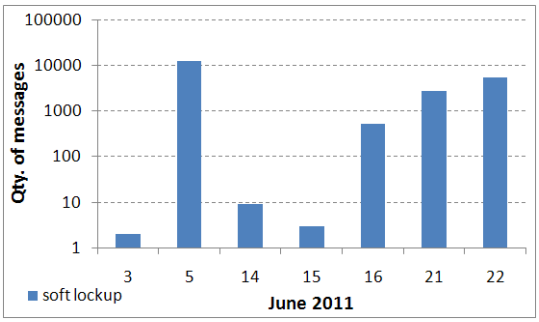
Table 3.1: List of resource use counters monitored on the Ranger, Lonestar4 and Stampede-1 HPC systems.

Metric group	**Qty.**	**Resource use counters**
Lustre network	6	tx_msgs, rx_msgs, rx_msgs_dropped, tx_bytes, rx_bytes, rx_bytes_dropped
Lustre /work, Lustre /share, Lustre /scratch	23 23 23	read_bytes, write_bytes, direct_read, direct_write, dirty_pages_hits, dirty_pages_misses, ioctl, open, close, mmap, seek, fsync, setattr, truncate, flock, getattr, statfs, alloc_node, setxattr, getxattr, listxattr, removexattr, inode_permission
Virtual memory	21	pgpgin, pgpgout, pswpin, pswpout, pgalloc_normal, pgfree, pgactivate, pgdeactivate, pgfault, pgmajfault, pgrefill_normal, pgsteal_normal, pgscan_normal, pgscan_direct_normal, pginodesteal, slabs_scanned, kswapd_steal, kswapd_inodesteal, pageoutrun, allocstall, pgrotated
Block md0, Block hdd	11 11	rd_ios, rd_merges, rd_sectors, rd_ticks, wr_ios, wr_merges, wr_sectors, wr_ticks, in_flight, io_ticks, time_in_queue
Cpu 0 to 15	112	user, nice, system, idle, iowait, irq, softirq
Mem 0 to 3	80	MemTotal, MemFree, MemUsed, Active, Inactive HighTotal, HighFree, LowTotal, LowFree, Dirty, Writeback, FilePages, Mapped, AnonPages, PageTables, NFS_Unstable, Bounce, Slab, HugePages_Total, HugePages_Free
Net ib0, Net lo, Net eth0,	23 23 23	collisions, multicast, rx_bytes, rx_compressed, rx_crc_errors, rx_dropped, rx_errors, rx_fifo_errors, rx_frame_errors, rx_length_errors, rx_missed_errors, rx_over_errors, rx_packets, tx_aborted_errors, tx_bytes, tx_carrier_errors, tx_compressed, tx_dropped, tx_errors, tx_fifo_errors, tx_heartbeat_errors, tx_packets, tx_window_errors
Numa 0 to 3	24	numa_hit, numa_miss, numa_foreign, interleave_hit, local_node, other_node
Ps	7	ctxt, processes, load_1, load_5, load_15, nr_running, nr_threads

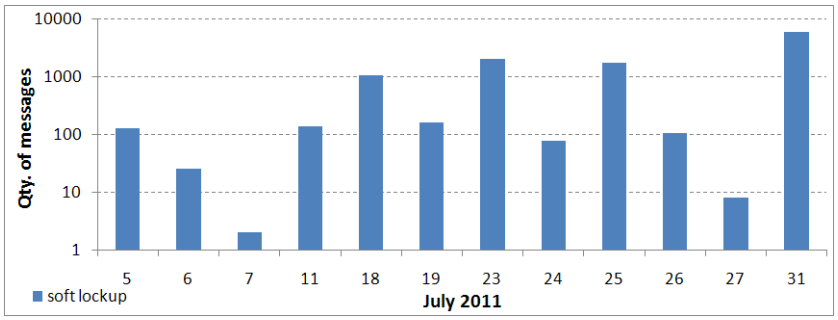
3.3.4 Compute Node Soft Lockups

Compute node lockups are one of the most frequent source of problems for the system administrators at the Texas Advanced Computing Center. A soft lockup is a bug which causes the Linux operating system kernel to loop without giving other tasks a chance to run. In a system message log, a soft lockup event can be identified by scanning the message for the keywords `soft lockup`. The soft lockups occur more than 1,000 times in one day. Figure 3.2 shows the number of soft lockups reported in three months worth of Rationalized message logs collected on the Ranger HPC system. Figure 3.3 shows the number of soft lockups reported in two months worth

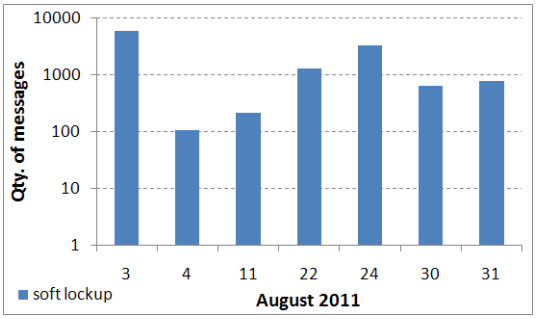
of Syslogs collected on the Lonestar4 HPC system.



(a) June 2011



(b) July 2011

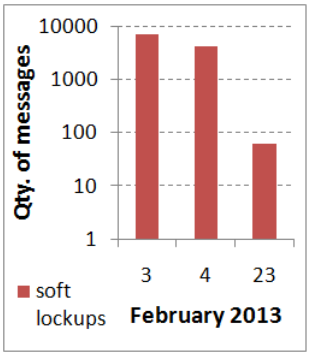


(c) August 2011

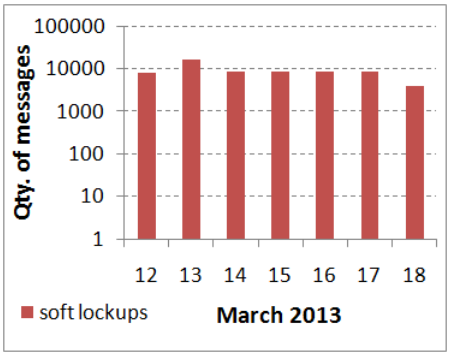
Figure 3.2: Distribution (log-scale) of soft lockup events on Ranger.

3.3.5 Data Collection

In Section 3.2, we reported that: (i) each node on Ranger, Lonestar4 and Stampede-1 generated its own resource use data, and (ii) each node on Ranger and Lonestar4 generated its own system logs. The Ranger, Lonestar4 and Stampede-1 HPC systems record 410 resource use counters across nine groups of system metrics (see Table 3.1). The resource use on Ranger, Lonestar4 and Stampede-1 were sampled at 10-minute intervals. The system logs generated on Ranger and Lonestar4 contain messages produced by the Linux operating system kernel, the Lustre filesystem and Linux



(a) February 2013



(b) March 2013

Figure 3.3: Distribution (log-scale) of soft lockup events on Lonestar4.

processes. There are no system message logs available on Stampede-1. A summary of the resource use data and system logs analysed in this thesis is given in Table 3.2.

Table 3.2: Summary of the data collected on Ranger, Lonestar4 and Stampede-1.

Ranger				
	Resource use data		**Rationalized message logs**	
Month	Size	Qty. lines	Size	Qty. messages
June 2011	29.8 GB	88,821,351	2.7 GB	10,021,516
July 2011	29.3 GB	92,425,427	9.6 GB	64,822,682
August 2011	29.9 GB	91,502,909	14.5 GB	114,745,476
Lonestar4				
	Resource use data		**Syslogs**	
Month	Size	Qty. lines	Size	Qty. messages
February 2013	24.9 GB	111,424,271	966 MB	8,993,154
March 2013	46.6 GB	207,068,692	1.3 GB	12,267,629
Stampede-1				
	Resource use data		**Syslogs**	
Month	Size	Qty. lines	Size	Qty. messages
February 2017	128.9 GB	236,509,583	N/A	N/A

3.4 Processing the Cluster Log-data

The CORRMEXT and EXERMEST diagnostics frameworks target processing of TACC_Stats resource use data, Rationalized message logs and Syslogs. The resource use data contain hundreds of different resource use counters. The system logs contain thousands of message types. The system message logs may be ambiguous and unstructured. To address the problem, we describe two data preprocessing modules. The modules are: (i) Resource Use Extractor, (ii) Message Types Extractor.

3.4.1 Resource Use Extraction Module

Currently, the Resource Use Extractor (*RUExt*) module extracts resource use counters from TACC_Stats resource use data. We present the resource use counters in a form on which standard analysis algorithms can be applied. *RUExt* generates a data matrix $DR_{timebins}$ that contain counts of resource use counters by time-bins. The time-bins are one hour, 30 minutes and 10 minutes. In a resource use counter data matrix, each row represents a resource use counter name, each column represents one time-bin and each cell contains the count of a resource use counter name within the time-bin. Currently, *RUExt* generates three different types of resource use counter data matrices of one hour, 30 and 10 minute time-bins. We generate the resource use counter data matrices using a process. The process is given below:

- Step 1: Split one day worth of resource usage logs into individual hours.
- Step 2: Split the hourly resource usage logs into 30 minutes.
- Step 3: Split the hourly resource usage logs into 10 minutes.
- Step 4: For each log entry in the individual hour resource usage logs, extract the resource use counter name and store it in a list.
- Step 5: Identify the unique resource use counter name in the list and obtain the list of resource use counters.
- Step 6: For each resource use counter name in the hourly resource usage log which matches the resource use counter name in the given list of resource use counter names, if the values associated with the resource use counter name of two consecutive hourly resource usage logs are different, obtain the difference and add the difference to the value obtained in the preceding operation and store the value.
- Step 7: Repeat Step 6 for the 30 minutes resource usage logs.
- Step 8: Repeat Step 6 for the 10 minutes resource usage logs.

3.4.2 Message Types Extraction Module

The Message Types Extractor (*MTExt*) module extracts message types from large quantities of system logs. We define a message type as a sequence of words that contain only alphabets in the English language. *MTExt* extracts message types from the error message part of a system log and presents the message types in the form of a data matrix that contain counts of message types by time-bins. The time-bins are

one hour, 30 minutes and 10 minutes. Currently, *MTExt* generates three different message types data matrices of one hour, 30 and 10 minute time-bins. In a message types data matrix, each row represents a message type, each column represents one time-bin and each cell contains the count of a message type within the time-bin. The data matrix provides a form on which standard analysis algorithms can be applied. We generate the message types data matrices using a process. The process is given below:

- Step 1: Split one day worth of system logs into logs of individual hours.
- Step 2: Split the hourly system logs into logs of 30 minutes.
- Step 3: Split the hourly system logs into logs of 10 minutes.
- Step 4: For each log message in the hourly system logs, extract the message type part and store it in a list.
- Step 5: Identify the unique message type in the list and obtain the list of message types.
- Step 6: Given a list of message types obtained on one day worth of system logs, count the number of message types by hour and obtain the hourly message types data matrix.
- Step 7: Given a list of message types obtained on one day worth of system logs, count the number of message types using the 30 minutes system logs and obtain the 30 minutes message types data matrix.
- Step 8: Given a list of message types obtained on one day worth of system logs, count the number of message types using the 10 minutes system logs and obtain the 10 minutes message types data matrix.

3.5 Summary

In this chapter, we described the system model, fault model and system issues, introduced the Ranger, Lonestar4 and Stampede-1 HPC systems operated by the Texas Advanced Computing Center at The University of Texas at Austin, described the TACC_Stats resource use data, Rationalized message logs and Syslogs, and gave the implementation details for the Resource Use Data and Message Types Data preprocessing modules.

In Chapters 4, 5 and 6, we present two system diagnosis workflows that are based on the system model, fault model and system issues described in this

chapter. In Chapter 4, we present the CORRMEXT framework to identify frequently occurring error cases. In Chapter 5, we show that CORRMEXT generalises on multiple HPC systems. In Chapter 6, we present the EXERMEST framework that uses the resource use data matrices and message types data matrices to identify rare error cases. We apply EXERMEST on multiple HPC systems.

Chapter 4

A Correlation-based Workflow for HPC Systems Diagnosis

Combining system logs with resource utilisation data have been shown to increase the accuracy of failure diagnosis. Most of the work on failure diagnosis have focused on identifying errors that lead to system failures only, but there is little work that study errors which lead to a system failure or recovery on real data.

We structure this chapter as follows: In Section 4.1, we introduce the CORRMEXT (**COR**relating **R**esource use data and **M**essage logs and **EX**tracting **T**imes) framework. In Section 4.2, we illustrate the system issue and describe the problem specification. In Section 4.3, we present the details of the Data Type Extractor, Correlation and Time-bin Extraction modules. In Section 4.4, we present the analyses from CORRMEXT through five error cases identified on the Ranger HPC system and conclude with a summary and a recommendation in Section 4.5.

4.1 Introduction

There are many frameworks that have shown that combining system logs with resource utilisation data increases the accuracy of error detection [33, 34] and failure diagnosis [8, 9, 72]. The diagnostics framework developed by Z. Zheng et. al. [72] uses RAS and job logs to identify application and hardware failures and in [71] they show that interesting failure characteristics can be identified by combining job and RAS logs. The ANCOR framework [8] uses resource usage data to identify anomalous nodes and then uses system logs to diagnose the cause of system failures. The CRUMEL framework [9] identifies correlations of resource use counters and errors to system failures. The approach developed by N. Gurumdimma et. al. [33] shows that the error handling time window can be increased by combining system

logs with resource usage data. CRUDE [34] shows that error detection accuracy can be increased by 85% over current state-of-the-art error detection techniques. The diagnostics frameworks reported in references [8, 9, 72] correlate errors to system failures only, but there is little work which study errors leading to a system failure or recovery.

Knowing when an error leads to a system failure is important. Consider the following diagnosis: (1) *"An inode on the Lustre filesystem failed which led to a communication error between the Lustre client and server"*, and (2) *"The Lustre client failed to communicate with the Lustre server"*. If diagnosis number 2 is only available to the system administrator, they may decide to change the Lustre filesystem configuration and increase the number of retries. However if diagnosis number 1 is also available, then the system administrator may decide against increasing the number of retries and choose to terminate the job because an inode failure cannot be recovered. Diagnosis number 1 provides more information about the underlying cause of the communication error, i.e., a corrupted inode. Thus, detailed knowledge about the error can help the system administrator make the right decision to solve the problem.

In this chapter, we introduce a new framework to provide more detailed system diagnosis. We name the framework CORRMEXT. CORRMEXT combines a resource monitoring system called TACC_Stats [17] and system logs based on message-log rationalization [36]. CORRMEXT identifies and links groups of resource use counters and system events on a given date. It applies multiple correlation algorithms. We implement a three-phase approach where: (i) CORRMEXT extracts groups of correlated resource use counters on the resource use data, (ii) CORRMEXT extracts groups of correlated messages on the system logs, and (iii) CORRMEXT extracts the variance on the time-bins of the groups of correlated resource use counters and groups of correlated messages. We show that CORRMEXT provides a pathway to the root-cause of successful and failed error recovery mechanisms which alternative diagnostics tools can not provide.

4.1.1 Contributions

In this chapter, we make the following contributions:

- We design, implement and evaluate a new framework that combines resource utilisation data with system logs for detailed HPC systems diagnosis. We name this framework CORRMEXT.
- We demonstrate that CORRMEXT can: (i) identify frequently occurring error cases, and (ii) report the success and failure of error recovery protocols.

- We show that more dates of groups of correlated resource use counters and groups of correlated errors can only be identified by applying multiple correlation algorithms.
- We include a detailed statistical validation step to ensure accurate system diagnosis. We show that all the correlations are statistically significant by applying the Bonferroni correction.
- We show that both the correlations of resource use counters and correlations of errors are required for identifying the earliest times of change in the system behaviour on all dates.

4.2 System Issue and Problem Specification

In this section, we illustrate the system issue and describe the problem specification.

4.2.1 System Issue

In Figure 4.1, we illustrate resource use counters and error messages by time.

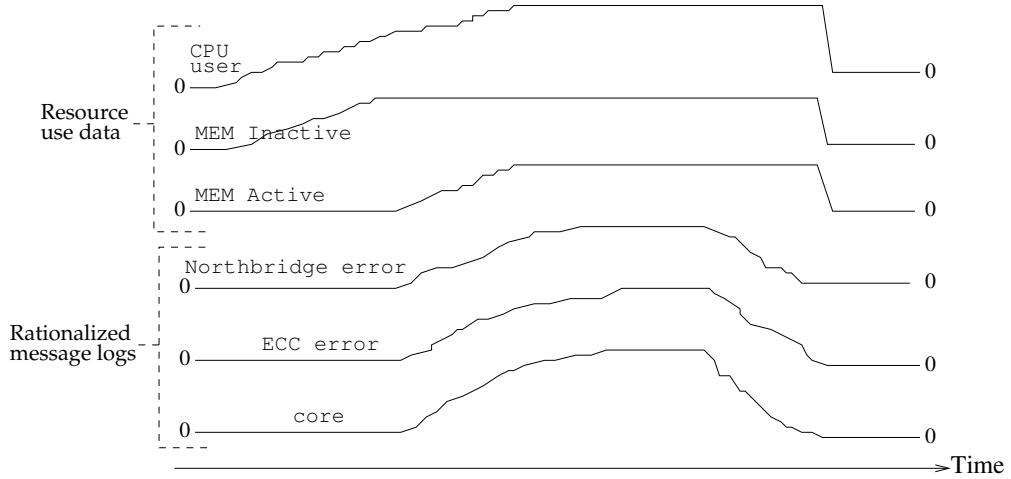


Figure 4.1: An illustration of the resource use counters: (i) user processor utilisation, (ii) memory pages not accessed recently and (iii) memory pages accessed recently, and the system messages: (i) northbridge error, (ii) ECC memory error and (iii) processor core.

`CPU user`, `MEM Inactive` and `MEM Active` are resource use counters. `CPU user` records CPU usage by a user. `MEM Inactive` records the number of memory pages that are not accessed recently. `MEM Active` records the number of memory pages that are accessed recently. We observe that `CPU user` and `MEM Inactive` are correlated. We observe that `CPU user` and `MEM Active` are correlated. `Northbridge Error`,

`ECC error` and `core` are system messages. `Northbridge Error` indicates a motherboard chipset error. `ECC error` indicates a ECC memory error. `core` indicates a processor core message. We observe that `Northbridge Error`, `ECC error` and `core` are correlated.

In the above example, an error in the system may be captured by a resource use counter r_1 and a recovery procedure may be captured by another resource use counter r_2. When r_1 and r_2 are correlated, it shows that the error triggered a recovery procedure. Similarly, when groups of different events are correlated, the correlations provide an insight into the state of the system. Therefore, to provide an insight into the system behaviour it is important to capture these correlations. However, the system logs provide low coverage. By low coverage, we mean that the system logs do not contain all the information required for establishing a causal path to the failure. In this chapter, we present the CORRMEXT framework that combines resource utilisation data with system logs to determine the occurrence of these patterns.

4.2.2 Problem Specification

The problem that we address in this chapter is specified in [10] and we describe the problem as follows: Given (i) a set of resource use data, (ii) a set of system logs, (iii) a list of resource use counter names, (iv) a list of message types, (v) a failure event, and (vi) a list of dates, then:

1. Identify groups of resource use counters that are strongly correlated by time-bins on the specified dates,
2. Identify groups of errors that are strongly correlated by time-bins on the specified dates,
3. Identify errors that are:
   (a) Strongly correlated to a specified failure event,
   (b) Weakly correlated to the specified failure event,
4. Identify the time-bins that are associated with the correlated resource use counters and correlated error groups on the dates specified.

The date specified captures the date where deeper insights are sought. To achieve this, we have developed the *CORRMEXT* (**COR**relating **R**esource use data and **M**essage logs and **EX**tracting **T**imes) framework as shown in Figure 4.2. The CORRMEXT framework is composed of three modules. The modules are: (1)

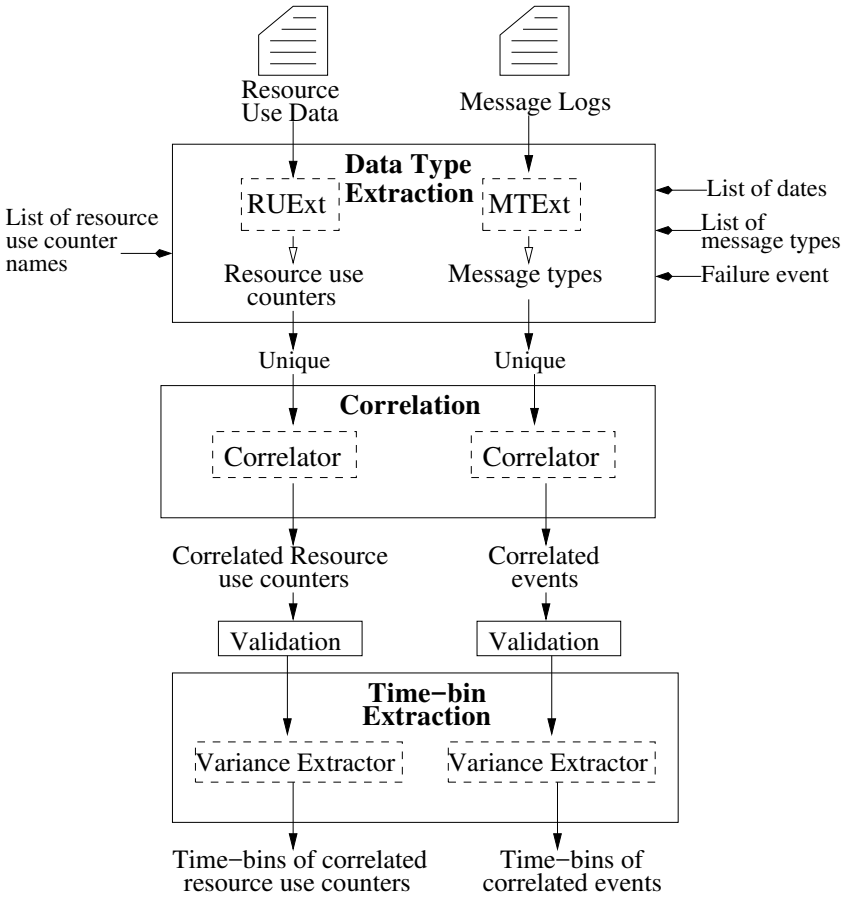


Figure 4.2: The workflow of the CORRMEXT framework.

Data Type Extraction, (2) Correlation and (3) Time-bin Extraction. The workflow automatically process the resource use and system logs through the Data Type Extraction, Correlation and Time-bin Extraction modules. Each module produces a report. The reports can be used for diagnosis. CORRMEXT is available for download at `https://diag-toolkits.github.io/CORRMEXT/`.

In the next section, we describe in detail the Data Type Extraction, Correlation and Time-bin Extraction modules used within the CORRMEXT framework.

4.3 CORRMEXT Framework

We have implemented a failure diagnostics framework called CORRMEXT to identify error cases that occur frequently and to report the success and failure of error recovery protocols. The framework is comprised of three-phases and we summarise it here. The three-phases are: (i) identifying resource use counter groups that are correlated, (ii) identifying error groups that are correlated, and (iii) identifying the earliest hour

of change associated with the correlated resource use counter groups and correlated error groups. Then, we validate that all the correlation coefficients of the correlated resource use counters and correlated errors are significant.

4.3.1 Data Type Extraction

Currently, the Data Type Extractor (*DTE*) module processes TACC_Stats resource use data [17], Rationalized message logs [36] and Syslogs [40]. TACC_Stats [17] is an online job-oriented system resource usage monitor. It monitors 410 resource use counters online and records all the resource use counters at intervals of 10-minutes (refer to Table 3.1 for the list of system metrics). When a node is reset, the readings on the node will be set to zero. An example of a resource use log is given as follows: `2066522 Aug 11 12:50:01 i150-412 eth0 rx_bytes 302345 ...` The resource use log contains: (i) `2066522` it is the job number, (ii) `Aug 11 12:50:01` it is the date and time, (iii) `i150-412` it is the node, (iv) `eth0` it is the component identifier and (v) `rx_bytes 302345` it is a key-value pair. The key-value pair contains the name of the resource use counter and its value.

The Rationalized message log [36] is a special type of system log that incorporates a logical structure and additional content such as job-identification. An example of a Rationalized message log is given as follows: `2055415 Aug 3 00:00:03 i120-306 kernel GSIFTP: failed`. The Rationalized message log contains: (i) `2055415` it is the job number, (ii) `Aug 3 00:00:03` it is the date and time, (iii) `i120-306` it is the node, (iv) `kernel` it is the software identifier and (v) `GSIFTP: failed` it is the message. A message is a sequence of English-only words. Here, the message is a GSIFTP failure.

A Linux syslog contains a date and time, node, software identifier and message. A Linux syslog typically does not contain a job number. Furthermore, the format of a system log on another HPC system may be different. Having said that, most system logs contain three basic fields. The fields are: (i) date and time, (ii) node and (iii) message. Therefore, we have implemented a log-reformator to convert system logs that contain the three basic fields to a standard format. If a system log does not contain a field in the standard formatted log, then a placeholder is used. The standard formatted log contains the following fields:

`job number, month, day, time-stamp (Hour:Minute:Second), node, software identifier, application name, message.`

The resource use data contain hundreds of different resource use counters. The Rationalized message logs and Syslogs contain thousands of different message types. Further, the resource use counters and system logs are collected at different

times. For example, the system resources used by a job are recorded at a regular time interval while a message is recorded in the system log only when an error is reported by the job. Therefore, we need a standardise way to represent the resource use counters and message types by time. To address the problem, we implemented two sub-modules within the Data Type Extractor. The sub-modules are: (i) a Resource Use Extractor (*RUExt*) and (ii) a Message Types Extractor (*MTExt*). The Resource Use Extractor organises the resource use counters. It receives a resource use log-file and outputs a data matrix $DR_{timebins}$. The data matrix contains counts of resource use counters by time-bins of one hour. The Message Types Extractor organises the message types. It receives a standard form log-file and outputs a data matrix $DM_{timebins}$. The data matrix contains counts of message types by time-bins of one hour.

4.3.2 Correlation

The Correlation module receives the time-bin data matrices that were generated by the Data Type Extractor. The Correlation module computes:

- The correlation coefficients for all the resource use counters and extracts a smaller set of resource use counters for analysis.
- The correlation coefficients for all the message types and extracts a smaller set of messages for analysis.

Our Correlation module currently evaluates two correlation methods. They are: (i) Pearson correlation, and (ii) Spearman-Rank correlation. The TACC_Stats system resource use monitor monitors all the instantaneous values for all system metrics at all time intervals. Any change in the resource use counter readings can be tracked by summing up all changes between consecutive time intervals. When the counts for a pair of resource use counters increase gradually, the Spearman-Rank correlation method can be used for capturing a monotonically increasing relationship between the resource use counters. The messages in the system logs are only instantaneous values. The counts for a pair of messages may increase gradually or fluctuate over time. Hence, the Pearson and Spearman-Rank correlation methods can be used for capturing message patterns which change over time. We use Pearson correlation to identify linear patterns of resource use counters and messages. We use Spearman-Rank correlation to identify monotonically increasing patterns of resource use counters and messages. Other methods are available. However, those methods assume that the variables in the data are independent and identically distributed. Therefore, the Pearson and Spearman-Rank correlation algorithms are

suitable methods. We implemented the Pearson and Spearman-Rank correlation methods into the Correlation module.

The Pearson correlation algorithm [67] assumes that the relationship between the data of two variables is linear and a line of best fit is drawn through the data of the two variables. Pearson's correlation coefficient, r is defined as the mean of the products of the standard scores, i.e.,

$$r = \frac{1}{n-1} \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{s_x} \right) \left(\frac{y_i - \bar{y}}{s_y} \right) \tag{4.1}$$

where $\left(\frac{x_i - \bar{x}}{s_x}\right)$ is the standard score of x, $\left(\frac{y_i - \bar{y}}{s_y}\right)$ is the standard score of y, x and y are two datasets containing n values of a pair of resource use counters or a pair of events, $s_x = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2}$ is the sample standard deviation of x, $s_y = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (y_i - \bar{y})^2}$ is the sample standard deviation of y, $\bar{x}$ and $\bar{y}$ is the sample mean of x and y.

The Spearman-Rank correlation algorithm [67] assumes that the relationship between the data of two variables is monotonic. A monotonic relationship does one of the following: (i) when the value of one variable increases the value of the other variable increases, (ii) when the value of one variable remains the value of the other variable remains. Spearman-Rank's correlation coefficient, ρ is defined as the Pearson correlation coefficient between a pair of ranked variables. To rank the variables, we implemented a standard technique that is called the tied rank average method [67]. We describe the ranking process as follows:

- Rank order the values in the dataset x with the smallest value getting a rank of 1.
- If more than one value has the same rank in dataset x, assign the average rank to these values.
- Rank order the values in the dataset y with the smallest value getting a rank of 1.
- If more than one value has the same rank in the dataset y, assign the average rank to these values.

The resource use data is generated at regular time intervals and the system logs are generated only if an error output statement in the code is triggered. Because of this, the resource use logs and system logs contain timestamps that are different. As a result, different numbers of data points are contained in the resource use counters and message types datasets. Both Pearson and Spearman-Rank correlation

algorithms require the same number of data points on the x-axis in the datasets. As such, we do not correlate a resource use counter and message type. Having said that, our objective is to identify correlations of resource use counters and correlations of system events. Therefore, we apply the Correlation module separately to the resource use data and system logs as shown in Figure 4.2.

After we have obtained the correlation matrices, we generate the lists of strong positive correlated resource use counters and strong positive correlated system events. We have configured the Correlation module using a process given in [8] and we describe the process as follows: $\forall r_{<m,n>} \in MTX_r$ where MTX_r is a correlation matrix, $r_{<m,n>}$ is a correlation coefficient in the correlation matrix MTX_r, m is the correlation matrix row index, n is the correlation matrix column index, if there are the most number of correlations such that $r_{<m,n>}$ of each correlation is the same, and $r_{<m,n>}$ lies between 0.8 and 1 (inclusive), then $r_{th} = r_{<m,n>}, m \neq n$. We use the following rules to interpret the strength of the correlation coefficient. The rules are [1]: (a) 0.8 to 1: Strong positive correlation, (b) 0.3 to 0.79: Moderate positive correlation, (c) 0.1 to 0.29: Weak positive correlation.

Our Correlation module extracts: (i) the resource use counters that are strong positive correlated, and (ii) the system events that are strong positive correlated. However, we also need to know if the system has recovered from an error. Because of this, the Correlation module extracts system events that are weakly correlated to a system failure event.

Validation

We test the significance of all the correlation coefficients by applying a standard technique called Fisher's z-transform. Fisher's z-transform is given by the equation [67]:

$$F(r) = \frac{1}{2} \ln \left(\frac{1+r}{1-r} \right) \tag{4.2}$$

where r is a correlation coefficient. We have defined the null (H_0) and alternate (H_a) hypotheses as follows: (i) H_{0r} that a pair of resource use counters are weakly positive correlated, (ii) H_{ar} that a pair of resource use counters are strongly positive correlated, (iii) H_{0e} that a pair of system events are weakly positive correlated, and (iv) H_{ae} that a pair of system events are strongly positive correlated. Then, we obtain the z-score for all correlation coefficients. The z-score is given by the equation [67]:

$$z = \frac{F(r) - u_z}{SE} = (F(r) - F(H_0)) \times \sqrt{n-3} \tag{4.3}$$

where n is the number of time-bins, $SE = \frac{1}{\sqrt{n-3}}$ is the standard error, and under

the null hypothesis $u_z = F(H_0)$. When $z \geq 2.64$ at 99% confidence level, we reject the null hypothesis in favour of the alternate hypothesis. We are interested in: (i) resource use counters that are strong positive correlated, and (ii) system events that are strong positive correlated. For all the hypotheses, we use the significance level, $\alpha = 0.01$ and apply a one-sided test to obtain all the P-values. A P-value less than 0.01 indicates that it is highly unlikely the result would be observed under the null hypothesis.

Handling False Positive

When we are given d number of hypotheses, the probability to observe one significant result due to chance is $1-(1-P)^d$ where P is the p-value obtained from each test. If we need to test only one hypothesis and obtain a P-value of 0.01, then the probability that this is a false positive is 1%. If we need to test more hypotheses, for example we have 26 hypotheses to test and obtain a P-value of 0.01 for each test, then the probability that there is at least one false positive is $1-(1-0.01)^{26} = 1-0.99^{26} = 0.22$ or 22%. The Bonferroni correction accounts for the inflation in false positive [26]. The Bonferroni correction works as follows: For each test, we apply the Bonferroni correction on the unadjusted P-value to obtain an adjusted P-value. We multiply the unadjusted P-value by d and obtain the adjusted P-value.

Implementation of Significance Testing

We describe the process we implemented for testing the significance of all the correlation coefficients: We use *RUExt* and *MTExt* to generate the number of hours for each date of the logs and use the Data Type Extractor (DTE) to generate the number of dates. Given the total number of hours in each date of logs and the correlation coefficients, the validation sub-components in the Correlation module use the correlation coefficients and number of hours to compute Fishers z-scores for all correlation coefficients – the flow is shown in Figure 4.2. Then, we map all the z-scores to P-values by using a Z-table – the P-values obtained are the unadjusted P-values. We implemented the Z-table in the Correlation module. Then, we multiply the unadjusted P-value by the total number of dates and obtain the adjusted P-value for all hypotheses.

4.3.3 Time-bin Extraction

The Time-bin Extraction module receives the data matrices that contain the resource use counters that are strongly correlated and system events that are strongly correlated. We generated the data matrices by:

- Mapping the names in the list of strongly correlated resource use counters to the names of the resource use counters in the resource use counters data matrix, and obtain a smaller resource use counters data matrix.
- Mapping the names in the list of strongly correlated messages to the names of the message types in the message types data matrix, and obtain a smaller message types data matrix.

The Time-Bin Extraction module obtains the variance for the correlated resource use counters and correlated system events at every hour to identify the hour that has the highest variance. The variance is given by the equation [67]:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{4.4}$$

where n is the sample size of x, $\bar{x}$ is the sample mean of x. Our objective is to identify the earliest hour of change in the system behaviour. To achieve this, we implemented a process for extracting the variance associated with the time-bins. We describe the process as follows:

1. Store the variance for each time-bin in a list l_{var}.
2. Obtain the difference in the variance between two consecutive time-bins and store the difference in a list $l_{vardiff}$.
3. Sort $l_{vardiff}$ in descending order with the first element the largest difference in variance.
4. The time-bin that has the highest variance is the earliest hour of change in the system behaviour during the day.

4.4 Case Study: Ranger HPC System

In this section, we study error cases within the context of the Ranger HPC system and then we will apply CORRMEXT to identify error cases on Lonestar4 and Stampede-1 in Chapter 5. The Ranger HPC system was operated by the Texas Advanced Computing Center at The University of Texas at Austin from 2007 – 2013. On Ranger, we collected three months worth of resource usage data and Rationalized message logs. In the reference [36], J. Hammond et. al. found that there is a lead time of six hours to a compute node crash when an error is first reported on Ranger. A compute node crash can be identified by searching the system log-message for the keywords `soft lockup`. To extract the dates when compute nodes crashed, we

implemented a function to scan the system logs for soft lockup keywords. When we find a soft lockup message, we extract the date associated with the soft lockup message. In the Rationalized message logs, we identified 26 dates of soft lockup events. The dates of resource usage data and Rationalized message logs analysed are given in Table 4.1.

Table 4.1: List of dates of log-data analysed on Ranger.

Month	**Dates**
June 2011	3, 5, 14, 15, 16, 21, 22
July 2011	5, 6, 7, 11, 18, 19, 23, 24, 25, 26, 27, 31
August 2011	3, 4, 11, 22, 24, 30, 31

We obtain the diagnostics reports generated by CORRMEXT. The diagnostics reports contain the lists of correlated resource use counters and correlated events. In the reports, we identified five error cases on the Ranger HPC system. The error cases are: (i) memory allocation and memory leaks, (ii) communication and filesystem I/O errors, (iii) chipset and memory errors, (iv) file access and process errors, and (v) process errors and memory exhaustion. All the error cases are different. We summarise the error cases in Table 4.2.

Table 4.2: List of error cases identified on the Ranger HPC system.

Component	**Error**	**No. of dates**
NUMA & process memory allocation	Memory allocation & memory leaks	10
Lustre filesystem & Infiniband	Communication & Lustre I/O errors	11
Chipset & ECC memory	Chipset & memory errors	13
Linux virtual memory	File access & process errors	8
Linux virtual memory	Process errors & memory exhaustion	2

4.4.1 NUMA and Process Memory Allocation

In this error case, we determine: (i) correlations between NUMA and Linux process resource use counters, and (ii) correlations between application memory leaks. We use the correlations to diagnose NUMA memory allocation problems. Then, we assess the system reliability.

Phase 1: Correlated NUMA & Process Resource Use Counters

When a node runs out of memory pages (an error occurs) but a process makes a request for memory pages on that node, the resource use counter named `numa miss` is incremented. When a process requests memory pages on the out-of-memory node but ends up being allocated memory pages on another node (a recovery from the

error), the resource use counter named `numa foreign` is incremented. When a Linux process is created, the resource use counter named `ps processes` is incremented. When a context switch occurs between the CPUs, the resource use counter named `ps ctxt` is incremented. The resource use counters that record NUMA and Linux processes activities can be used to see what happens when a Linux process makes a request for memory on a node that has run out of free memory pages.

Figure 4.3 shows the correlations between the resource use counters NUMA miss, Linux process and context switch in June 2011.

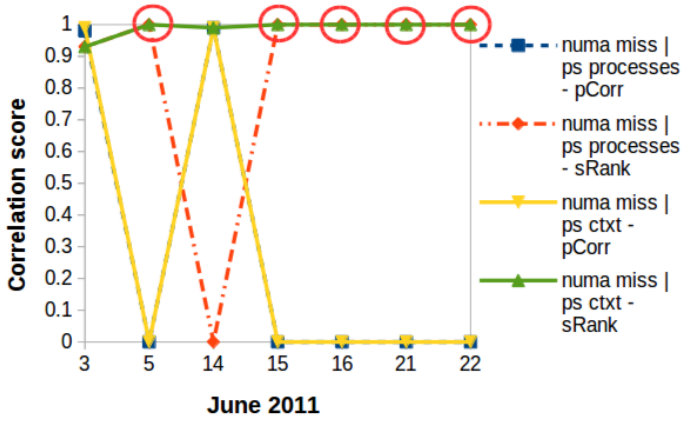


Figure 4.3: The full-circled counters were identified by Spearman-Rank correlation only.

Figure 4.4 shows the correlations between the resource use counters NUMA miss, Linux process and context switch in July 2011.

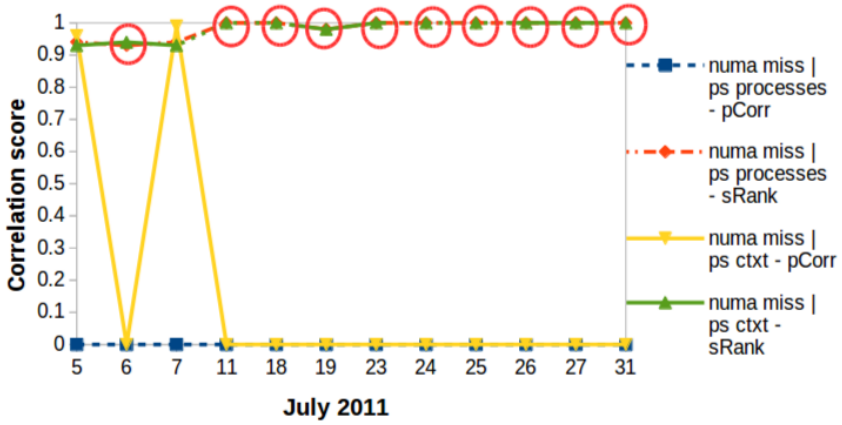


Figure 4.4: The full-circled counters were identified by Spearman-Rank correlation only.

Figure 4.5 shows the correlations between the resource use counters NUMA

miss, Linux process and context switch in August 2011.

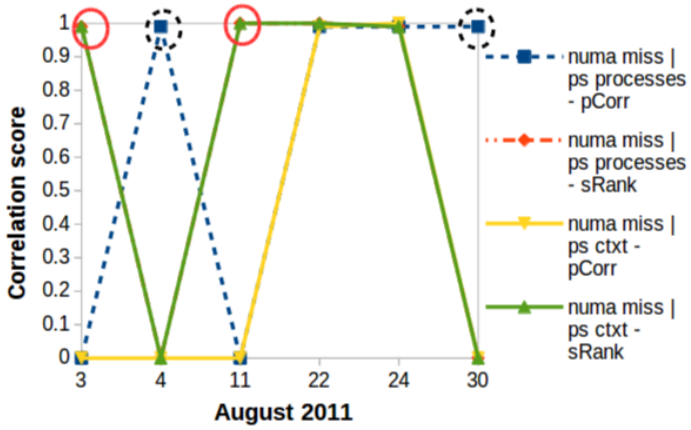


Figure 4.5: The full-circled counters were identified by Spearman-Rank correlation only. The dot-circled counters were identified by Pearson correlation only.

From Figure 4.3, Figure 4.4 and Figure 4.5, we observe there is a strong positive correlation between NUMA miss, Linux processes created and context switches. We identified: (i) correlations of `numa miss` and `ps processes` with scores ranging from 0.93 to 1 on 25 dates, and (ii) correlations of `numa miss` and `ps ctxt` with scores ranging from 0.93 to 1 on 23 dates. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one hour.

Figure 4.6 shows the correlations between the resource use counters NUMA foreign, Linux process and context switch in June 2011.

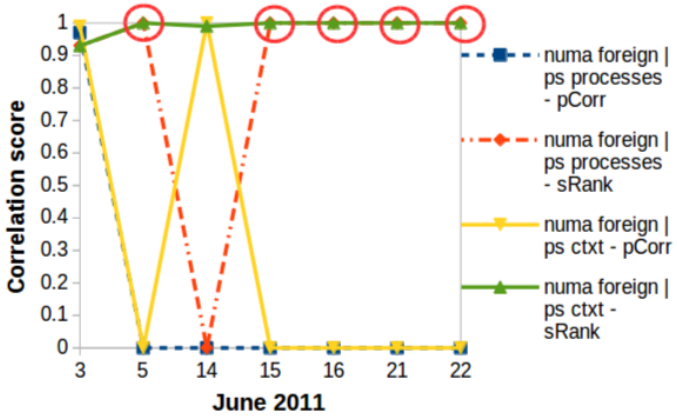


Figure 4.6: The full-circled counters were identified by Spearman-Rank correlation only.

Figure 4.7 shows the correlations between the resource use counters NUMA foreign, Linux process and context switch in July 2011.

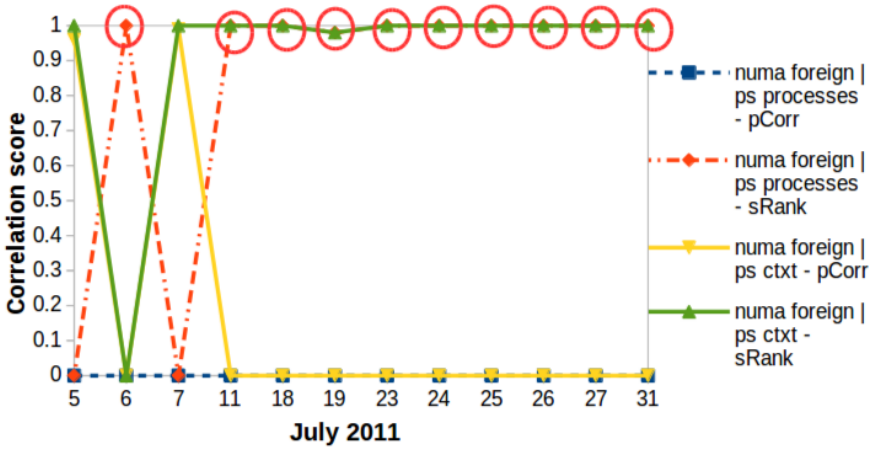


Figure 4.7: The full-circled counters were identified by Spearman-Rank correlation only.

Figure 4.8 shows the correlations between the resource use counters NUMA foreign, Linux process and context switch in August 2011.

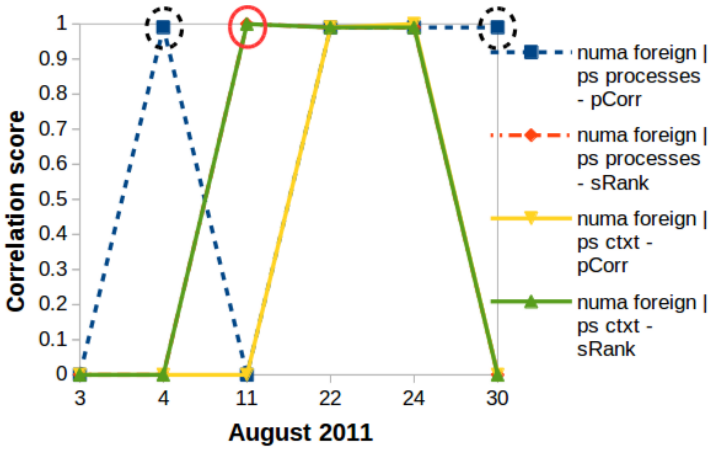


Figure 4.8: The full-circled counters were identified by Spearman-Rank correlation only. The dot-circled counters were identified by Pearson correlation only.

From Figure 4.6, Figure 4.7 and Figure 4.8, we observe that there is a strong positive correlation of NUMA foreign, Linux processes created and context switches. We identified: (i) correlations of `numa foreign` and `ps processes` with scores ranging from 0.93 to 1 on 21 dates, and (ii) correlations of `numa foreign` and `ps ctxt` with scores ranging from 0.93 to 1 on 21 dates. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation

scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one hour.

On August 4 and 30, we found that the correlated NUMA & processes resource use counters were identified only by Pearson correlation. However, we also found that on 17 dates the correlated NUMA miss & Linux processes resource use counters were identified only by Spearman-Rank correlation. On 16 dates the correlated NUMA foreign & Linux processes resource use counters were identified only by Spearman-Rank correlation. If we use only the Pearson correlation method, then the correlated NUMA & processes resource use counters would not be identified on more dates. If we use only the Spearman-Rank correlation method, then the correlated NUMA & processes resource use counters would not be identified on August 4 and 30. Our results show that:

- Both the Pearson and Spearman-Rank correlation methods are required. On August 4 and 30, correlations of NUMA memory allocation and Linux process resource allocations were identified only by Pearson correlation. On 16 dates, correlations of NUMA foreign and Linux process resource allocations were identified only by Spearman-Rank correlation. On 17 dates, correlations of NUMA miss and Linux process resource allocations were identified only by Spearman-Rank correlation.

- The system attempted a recovery when a process made a request for memory pages on a node that had run out of memory. The system recovers by allocating memory pages on another node for the process. We observed that context switching occurs when memory pages are allocated on another node.

In the second phase, we will use the correlations of two different groups of error events to diagnose application memory leaks.

Phase 2: Correlated Segmentation Faults & General Protection Errors

A program that attempts a read or write operation in a protected memory location will cause the operating system to report a segmentation fault. When a program accesses a memory location that is protected, a general protection fault (GPF) interrupt is issued by the processor. When a GPF is issued, the operating system removes the program, issues a signal to the user and continues executing other programs. In most cases, the operating system catches the general protection fault interrupt. However, in some cases the operating system may fail to catch the GPF. If the operating system fails to catch the GPF, the processor issues a second GPF. However, in a rare case the operating system may fail to catch the second GPF. If the operating system fails to catch the second GPF, the processor stops working and it will only

respond to a reset. A segmentation fault can be identified in a system message-log containing the keywords `segfault`. A memory access violation can be identified in a system message-log containing the keywords `general protection error`.

Figure 4.9 shows the correlation of segmentation fault and general protection error events in June 2011.

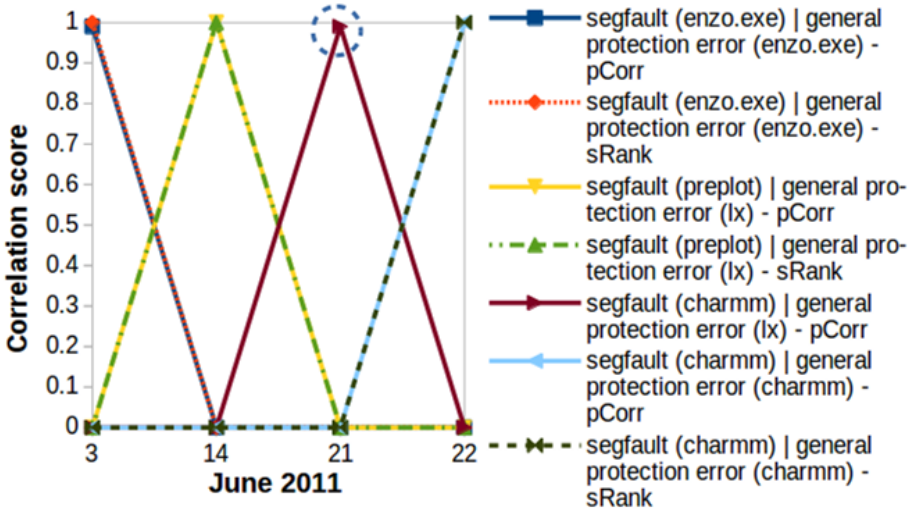


Figure 4.9: The dot-circled events were identified by Pearson correlation only.

Figure 4.10 shows the correlation of segmentation fault and general protection error events in July 2011.

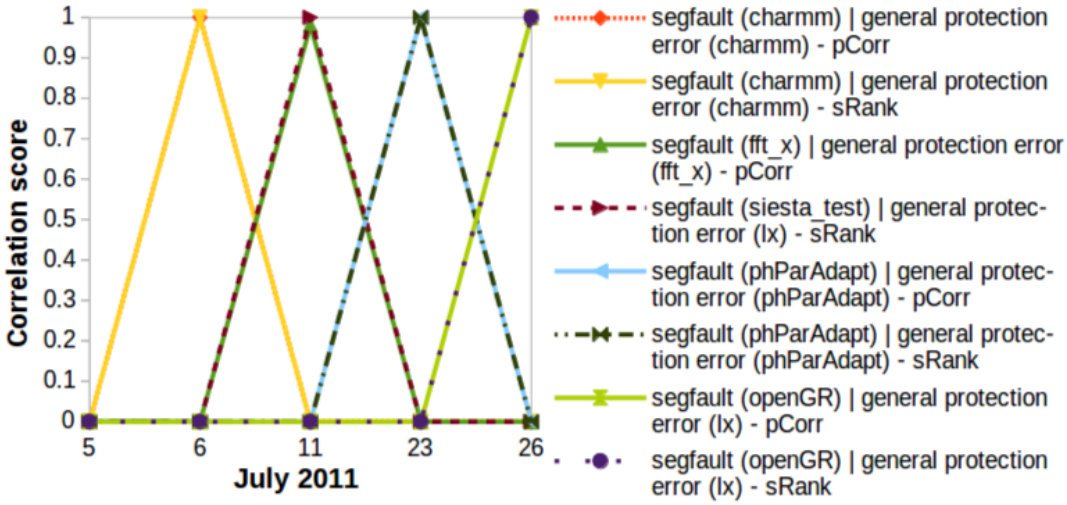


Figure 4.10: Correlations of segmentation fault and general protection error events.

Figure 4.11 shows the correlation of segmentation fault and general protection error events in August 2011.

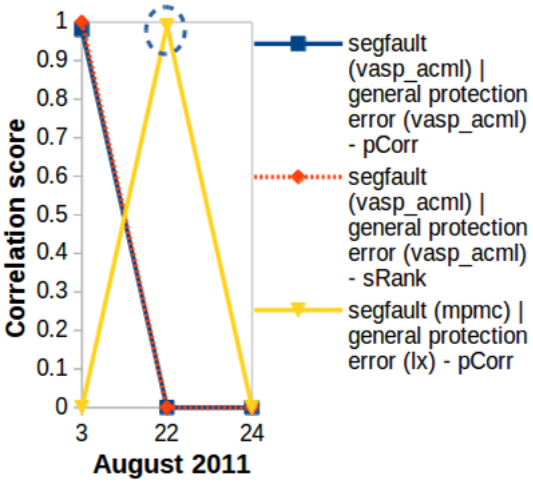


Figure 4.11: The dot-circled events were identified by Pearson correlation only.

From Figure 4.9, Figure 4.10 and Figure 4.11, we observe that there is a strong positive correlation between segmentation fault and general protection error events. On 10 dates, we identified correlations of `segfault` and `general protection error` events with scores ranging from 0.98 to 1. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one hour. The dates coincide with all the dates of correlated NUMA miss and Linux process resource use counters, and correlated NUMA foreign and Linux process resource use counters (see Section 4.4.1). We found that the correlated segmentation faults and general protection errors were identified only by Pearson correlation on June 21 and August 22.

Next, we identify which program caused the segmentation fault. To identify the faulty program, we implemented a function that scans the system message-log for the keywords `general protection error` and `segfault`. We obtained the segmentation fault and general protection error messages and extract the program name. The name of the program is appended at the back of the message. We identified 10 programs. The programs are: `enzo.exe`, `preplot`, `charmm`, `fft_x`, `siesta_test`, `lx`, `phParAdapt`, `openGR`, `vasp_acml` and `mpmc`. `enzo.exe` is an executable for a cosmology simulation program. `preplot` is used by analysis programs to reformat the output for plotting in GNUplot. `charmm` is the name for a molecular simulation program. `fft_x` is a fast Fourier transform algorithm. `siesta_test` is a unit testing tool for JavaScript. `phParAdapt` is a program that simulates multiphase flows using a parallel adaptive mesh method. `openGR` is a program that supports large numerical simulations in general relativity. `vasp_acml` is a Vienna Ab initio simulation package

and AMD core maths library. `mpmc` is a Massively Parallel Monte Carlo method package.

Correlations with failures: Next, we determine the correlation strength of: (i) `segfault` and `soft lockup` events, and (ii) `general protection error` and `soft lockup` events. We implemented a function to scan the list of correlated events and a summary is provided in Table 4.3.

Table 4.3: Summary of correlated "segfault" and soft lockup events, and correlated "general protection error" and soft lockup events on Ranger.

Error event	Failure event	Date	pCorr	sRank
segfault (charmm)	soft lockup	July 6	0.99	1
general protection error (charmm)	soft lockup	July 6	1	1
segfault (phParAdapt)	soft lockup	July 23	0.99	1
general protection error (phParAdapt)	soft lockup	July 23	0.99	1

From Table 4.3, we observe that: (i) `charmm` and `phParAdapt` programs are associated with both segmentation fault and general protection errors, and (ii) there is a strong positive correlation of `segfault` and `general protection error` events to soft lockup events on July 6 and 23. Our results show that memory access violation by the `charmm` and `phParAdapt` programs led to soft lockups on July 6 and 23. On the other eight dates, we found that the segmentation fault and general protection error events are weakly correlated.

Detailed diagnosis: During a system recovery that was caused by a NUMA miss, several programs violated the protected memory location policy and caused the Linux operating system to report a segmentation fault. On eight of ten dates the faulty program was removed by the operating system. This represents a recovery rate of 80%. However, on two dates the general protection fault triggered by two programs were not caught by the Linux operating system which led to failure.

The benefit of combining analysis of NUMA and Linux process resource use counters with program memory leaks is as follows: When on the same day, Linux process and NUMA resource use counters are correlated and segmentation fault and general protection error events are also correlated, it shows that memory errors are generated by memory allocation activities. Therefore, we can: (i) use the correlation of NUMA and Linux process resource use counters to monitor the state of memory allocation, and (ii) use the correlation of segmentation fault and general protection error events to identify the program that caused the memory leak.

Phase 3: Earliest Hour of Change

Table 4.4 shows the earliest hour of change in the correlated NUMA & Linux process resource use counters and correlated segmentation fault and general protection error events.

Table 4.4: Hours associated with the correlated NUMA & Processes resource use counters and correlated segmentation fault & general protection fault error messages on Ranger.

Correlated counters	**Jun 3**	**Jun 14**	**Jun 21**	**Jun 22**	**Jul 6**	**Jul 11**	**Jul 23**	**Jul 26**	**Aug 3**	**Aug 22**
NUMA & processes	1 PM	1 PM	3 AM	10 AM	12 PM	7 PM	5 AM	5 PM	1 PM	1 AM
Correlated errors	**Jun 3**	**Jun 14**	**Jun 21**	**Jun 22**	**Jul 6**	**Jul 11**	**Jul 23**	**Jul 26**	**Aug 3**	**Aug 22**
Segfaults & GPF errors	1 PM	7 AM	11 AM	5 AM	2 PM	11 PM	5 PM	7 PM	4 AM	10 AM

On all the dates, we observe that the earliest hour of change is different. On six dates, the earliest hour of change is associated with the correlated NUMA & Linux process resource use counters. On three dates, the earliest hour of change is associated with the correlated segmentation fault & general protection error events. On one date, the earliest hour of change is associated with both the correlated resource use counters and correlated errors. If we use only the correlated segmentation faults and general protection error events, the earliest hour of change on six dates would not be identified. If we use only the correlated NUMA & Linux process resource use counters, the earliest hour of change on three dates would not be identified. Our results show that we require both the correlated resource use counters and correlated errors for identifying the earliest hour of change in the system behaviour on all the dates. We found that there are different time-windows between the hours of change on all the dates. The time-window ranges from one hour to 12 hours.

Validation

Next, we test the significance of the correlation coefficient of: (i) groups of resource use counters that are strong positive correlated, and (ii) groups of error events that are strong positive correlated. We tested all the correlation coefficients against the null hypothesis. We obtained the z-scores for all the correlation coefficients and a summary is given in Table 4.5.

From Table 4.5, we observe that the z-scores for all the correlation coefficients range from 6.15 to 10.68. At the 99% confidence level, under the null hypothesis

Table 4.5: Summary of z-scores. n contains the number of hours in one day of logs.

Correlated groups	June 2011	July 2011	Aug 2011
NUMA & Processes resource use counters ($n = 24$)	$z_r = 10.68$	$6.15 \leq z_r \leq 10.68$	$z_r = 10.68$
Segmentation fault & GPF errors ($15 \leq n \leq 24$)	$z_e = 10.68$	$9.08 \leq z_e \leq 10.68$	$9.08 \leq z_e \leq 10.68$

$z_{0r} = 2.64$ and $z_{0e} = 2.64$. Hence, we reject the null hypothesis in favour of the alternate hypothesis.

Next, for all hypothesis tests we use the significance level, $\alpha = 0.01$ and apply a one-sided test to obtain a P-value. We use the P-value to determine the probability of rejecting the null hypothesis when it is true. Table 4.5 summarises the z-scores for all the correlation coefficients. We observe that the smallest z-score is 6.15. Since this is a one-sided test, the P-value is equal to the probability of observing a value greater than 6.15 in the standard normal distribution, or $P(Z > 6.15) = 1 - P(Z \leq 6.15) = 1 - 0.99999 = 0.00001$. To account for inflation in false positive due to multiple independent tests, we obtain the adjusted P-value $0.00001 \times 25 = 0.00025$ where 25 is the number of dates. The adjusted P-value is less than 0.01, indicating it is highly unlikely this result would be observed under the null hypothesis. From Table 4.5, we observe that all the z-scores are greater than or equal to 6.15. Therefore, the adjusted P-value for all the z-scores are less than 0.01, indicating it is highly unlikely these results would be observed under the null hypothesis.

4.4.2 Lustre Filesystem and Infiniband Network

In this error case, we determine: (i) correlations between the Infiniband network and Lustre filesystem resource use counters, and (ii) correlations between Lustre filesystem and Infiniband network communication errors. We use the correlations to diagnose problems on the Lustre filesystem and Infiniband network. Then, we assess the reliability of the Lustre filesystem inodes and Infiniband network.

Phase 1: Correlated Infiniband Network & Lustre Filesystem Resource Use Counters

When network packets are dropped on the Infiniband network, the resource use counter named `net ib0 tx_dropped` is incremented. When network packets are transmitted on the Infiniband network, the resource use counter named `net ib0 tx_packets` is incremented. When data is being read on the Lustre filesystem share partition, the resource use counter named `llite /share read_bytes` is incremented.

When data is being written to the Lustre filesystem share partition, the resource use counter named `llite /share write_bytes` is incremented. The Infiniband network packet drop and network packets transmitted, and Lustre filesystem read and write resource use counters can be used to see what happens when the network and filesystem are heavily used.

Figure 4.12 shows the correlations between the resource use counters network packet drop, filesystem read and write bytes in June 2011.

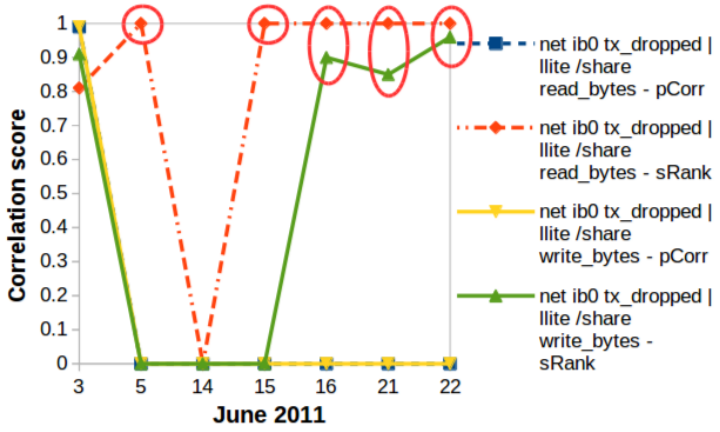


Figure 4.12: The full-circled counters were identified by Spearman-Rank correlation only.

Figure 4.13 shows the correlations between the resource use counters network packet drop, filesystem read and write bytes in July 2011.

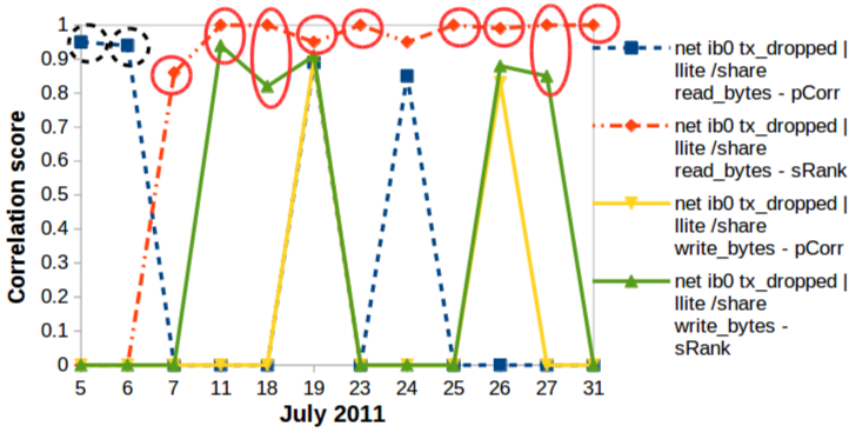


Figure 4.13: The full-circled counters were identified by Spearman-Rank correlation only. The dot-circled counters were identified by Pearson correlation only.

Figure 4.14 shows the correlations between the resource use counters network packet drop, filesystem read and write bytes in August 2011.

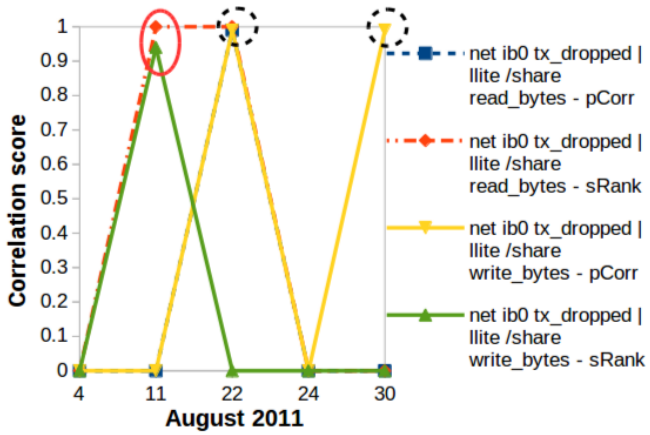


Figure 4.14: The full-circled counters were identified by Spearman-Rank correlation only. The dot-circled counters were identified by Pearson correlation only.

From Figure 4.12, Figure 4.13 and Figure 4.14, we observe that there is a strong positive correlation between: (i) network packet drop and filesystem read bytes, and (ii) network packet drop and filesystem write bytes. We identified: (i) correlations between `net ib0 tx_dropped` and `llite /share read_bytes` with scores ranging from 0.81 to 1 on 20 dates, and (ii) correlations between `net ib0 tx_dropped` and `llite /share write_bytes` with scores ranging from 0.82 to 0.99 on 12 dates. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one hour.

Figure 4.15 shows the correlations between the resource use counters network packets transmitted, filesystem read and write bytes in June 2011.

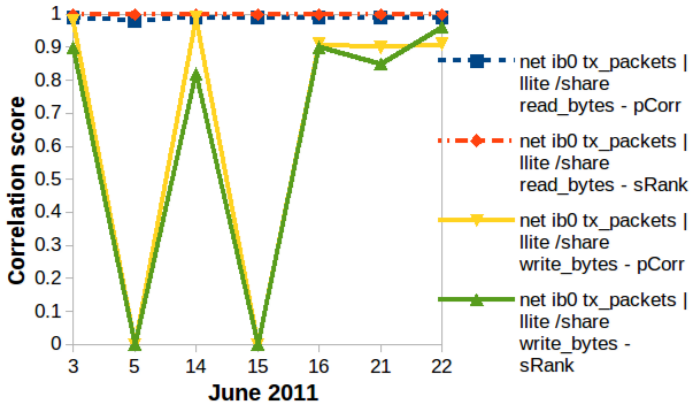


Figure 4.15: Correlation of network packets transmitted, filesystem read bytes and filesystem write bytes.

Figure 4.16 shows the correlations between the resource use counters network packets transmitted, filesystem read and write bytes in July 2011.

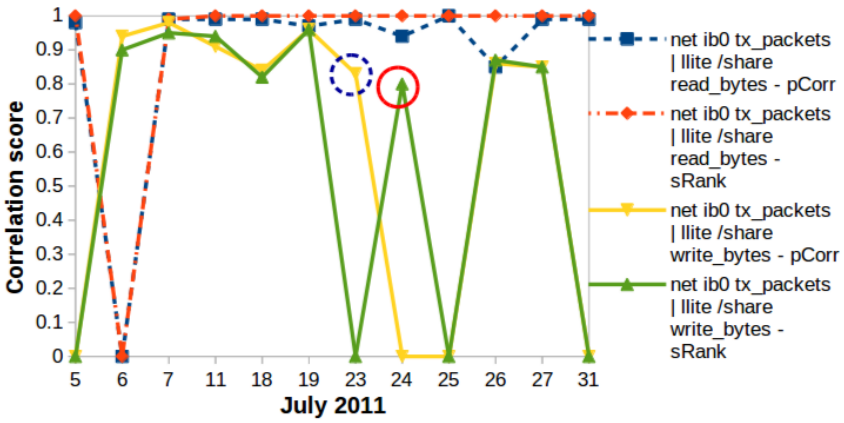


Figure 4.16: The full-circled counters were identified by Spearman-Rank correlation only. The dot-circled counters were identified by Pearson correlation only.

Figure 4.17 shows the correlations between the resource use counters network packets transmitted, filesystem read and write bytes in August 2011.

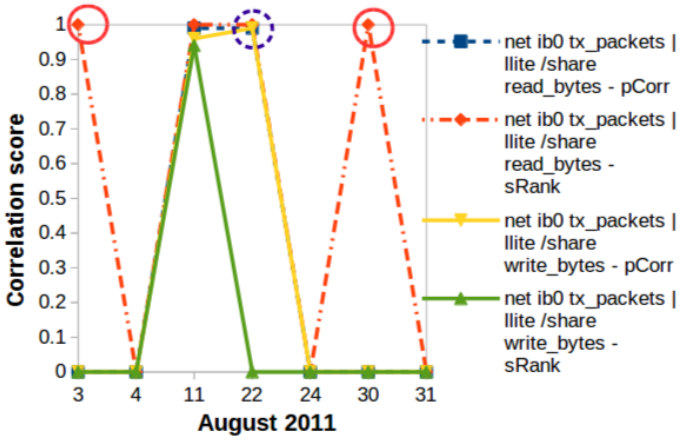


Figure 4.17: The full-circled counters were identified by Spearman-Rank correlation only. The dot-circled counters were identified by Pearson correlation only.

From Figure 4.15, Figure 4.16 and Figure 4.17, we observe that there is a strong positive correlation between: (i) network packets transmitted and filesystem read bytes, and (ii) network packets transmitted and filesystem write bytes. We identified: (i) correlations between `net ib0 tx_packets` and `llite /share read_bytes` with scores ranging from 0.94 to 1 on 22 dates, and (ii) correlations between `net ib0 tx_packets` and `llite /share write_bytes` with scores ranging from 0.80 to 0.99 on 16 dates. There are small changes in the correlation scores obtained

on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one hour.

On five dates, the correlated `net ib0 tx_dropped` and `llite /share read_bytes`, `net ib0 tx_dropped` and `llite /share write_bytes`, and `net ib0 tx_packets` and `llite /share write_bytes` were identified only by Pearson correlation. On 18 more dates, the correlated `net ib0 tx_packets`, `net ib0 tx_dropped`, `llite /share read_bytes` and `llite /share write_bytes` were identified only by Spearman-Rank correlation. If we use only the Pearson correlation method, then the correlated Infiniband and Lustre filesystem resource use counters on 18 dates would not be identified. If we use only the Spearman-Rank correlation method, then the correlated Infiniband and Lustre filesystem resource use counters on five more dates would not be identified. Our results show that:

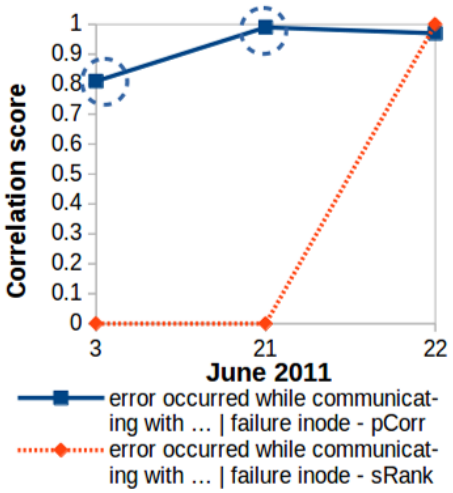
- There is a strong relationship of Infiniband network packets transmitted, Infiniband network packet dropped, Lustre filesystem read bytes and Lustre filesystem write bytes on 24 dates.
- Both the Pearson and Spearman-Rank correlation methods are required. On five dates, the correlated Infiniband network and Lustre filesystem activities were identified only by Pearson correlation. On 18 more dates, the correlated Infiniband network and Lustre filesystem activities were identified only by Spearman-Rank correlation.

In the second phase, we will use the correlations between two different groups of error events to diagnose communication and Lustre filesystem errors.

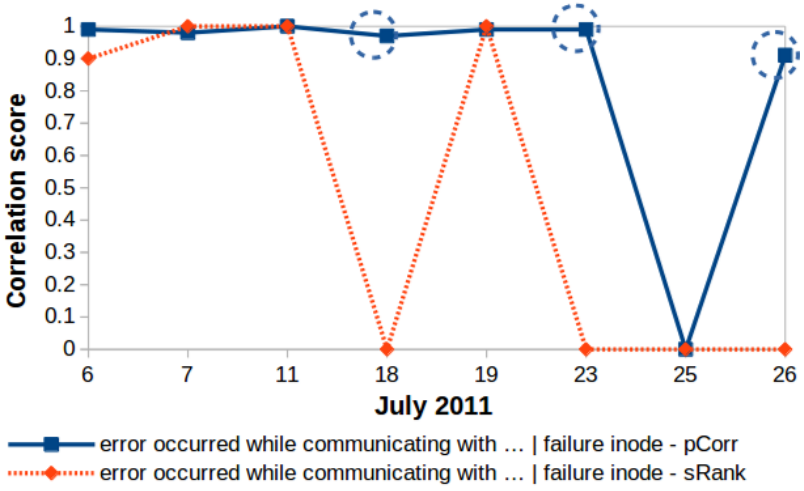
Phase 2: Correlated Communication & Lustre Filesystem Errors

In the system logs, we identify a communication error by searching for a message-log that contains the keywords `error occurred while communicating with ...`. We identify Lustre filesystem errors by searching the system logs for a message-log that contains either `failure inode` or `error reading dir`. In a Unix-style filesystem, the attributes about a file and the location of disk blocks where the file is stored are contained in a data structure called an inode. Inodes provide clients the information they need in order to access files that are stored in a storage server. When an inode is corrupted, the information stored in the inode is lost. If a client accesses a corrupted inode, it is unable to obtain the information for locating the file. A corrupted filesystem partition or faulty hard drive can result in inode failures.

Figure 4.18 shows the correlations between Infiniband network communication error and Lustre filesystem inode failure events in June and July 2011.



(a) June 2011.



(b) July 2011.

Figure 4.18: The dot-circled events were identified by Pearson correlation only.

From Figure 4.18, we observe that there is a strong positive correlation between Infiniband communication errors and Lustre inode failures. We identified: (i) correlations between `error occurred while communicating with ..` and `failure inode` events with scores ranging from 0.81 and 1 on 10 dates. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one hour.

Figure 4.19 shows the correlations between Infiniband network communication error and Lustre filesystem directory read error in August 2011. We observe that there is a strong positive correlation between Infiniband communication error and

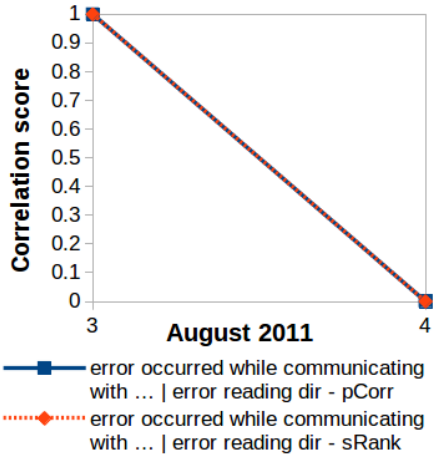


Figure 4.19: Correlation of communication error and directory read error events.

Lustre filesystem directory read error. We identified: (i) correlations between `error occurred while communicating with ..` and `error reading dir` events with a score of 1 on August 3. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one hour.

On 11 dates, we found that the dates of correlated communication and filesystem errors coincide with all the dates of correlated Lustre filesystem and Infiniband network resource use counters. On the 11 dates, we found that the correlated communication and filesystem errors were identified by Pearson correlation. However, the correlated communication and filesystem errors were identified by Spearman-Rank correlation only on six out of the 11 dates. Our results show that the correlated communication and filesystem errors were identified only by the Pearson correlation method.

To determine the cause of the communication error, we implemented a function to scan the `error occurred while communicating with ..` message for keywords `failed with Lustre` and `failed with client.c`. If the keywords `failed with Lustre` and `failed with client.c` are contained in the communication error message, then it shows that the communication error is associated with the Lustre filesystem. On 10 dates, we found that the `failed with Lustre` and `failed with client .c` keywords are contained in the `error occurred while communicating with ..` messages. The communication error message is correlated to `failure inode` on all the 10 dates. This shows that the communication error was caused by an inode failure.

Correlations with failures: Next, we determine the correlation strength between communication errors, filesystem inode failures and soft lockup events. We implemen-

Table 4.6: Summary of correlated "error occurred while communicating with" and "soft lockup", and correlated "failure inode" and "soft lockup" events on Ranger.

Error event	Failure event	Date	pCorr	sRank
error occurred while communicating with	soft lockup	June 21	1	-
failure inode	soft lockup	June 21	1	-
error occurred while communicating with	soft lockup	July 23	0.99	-
failure inode	soft lockup	July 23	0.99	-

ted a function to scan the list of correlated events for the `error occurred while communicating with ...`, `failure inode` and `soft lockup` messages. Table 4.6 provides a summary of the correlated events. From Table 4.6, we observe that: (i) there is a strong positive correlation between communication errors and soft lockup events on June 21 and July 23, and (ii) there is a strong positive correlation between inode failure and soft lockup events also on June 21 and July 23. We found that the communication error, inode failure and soft lockup events were weakly correlated on eight dates.

Detailed diagnosis: On 10 dates in June and July 2011, some inodes on the Lustre filesystem became corrupted. Because of this, the remote client was unable to access a file on Lustre. On 8 of 11 dates, the inode failure did not cause the remote client to hang. This represents a recovery rate of 72%. However, on 2 of 11 dates the inode failure led to a remote client hang, representing a failure rate of 18%.

We found that there is a strong positive correlation between `error occurred while communicating with ...` and `error reading dir` events – the correlation score is 1 (see Figure 4.19). We scanned the `error occurred while communicating with ...` message and identified the keywords `failed with Lustre` and `failed with client.c` in the message. We scanned the list of correlated events to determine the correlation strength between the communication error, error reading directory and soft lockup events. We found that `error occurred while communicating with ...` and `error reading dir` events were weakly correlated to soft lockup event.

Detailed diagnosis: On August 3, a remote client made a directory access request on the Lustre filesystem. When the client attempted to read the directory, the Lustre filesystem generated a directory read error and communicated the error to the remote client. The directory read error did not cause the remote client to hang.

Combining analysis of Lustre filesystem and Infiniband network resource use counters and errors provides the following benefit: When on the same day, we observe: (i) Lustre I/O and Infiniband network resource use counters are strongly positive correlated, and (ii) Lustre filesystem and communication error events are strongly positive correlated, it shows that Lustre filesystem and communication errors were

generated by Lustre I/O and Infiniband network activities. Therefore, we can use the correlations between Lustre I/O and Infiniband network resource use counters to monitor the state of Lustre and Infiniband.

Phase 3: Earliest Hour of Change

Table 4.7 provides the earliest hour of change in the correlated Lustre I/O and Infiniband resource use counters and correlated Lustre filesystem and communication errors.

Table 4.7: Hours associated with the correlated Infiniband and Lustre I/O resource use counters and correlated communication and Lustre filesystem errors on Ranger.

Correlated counters	**Jun 3**	**Jun 21**	**Jun 22**	**Jul 6**	**Jul 7**	**Jul 11**	**Jul 18**	**Jul 19**	**Jul 23**	**Jul 26**	**Aug 3**
Infiniband & Lustre I/O	12 PM	10 AM	11 AM	12 PM	12 PM	11 PM	7 AM	1 PM	7 AM	6 AM	11 AM
Correlated errors	**Jun 3**	**Jun 21**	**Jun 22**	**Jul 6**	**Jul 7**	**Jul 11**	**Jul 18**	**Jul 19**	**Jul 23**	**Jul 26**	**Aug 3**
Communication & Lustre FS	8 PM	10 PM	5 PM	11 PM	12 PM	11 PM	10 PM	9 PM	6 PM	3 PM	3 AM

On all the dates, we observe that there are different hours of change. On eight dates, the earliest hour of change is associated with the correlated Infiniband and Lustre I/O resource use counters. On one date, the earliest hour of change is associated with the correlated communication and Lustre filesystem errors. On two dates, the earliest hour of change is associated with both the correlated resource use counters and correlated errors. If we use only the correlated error events, then the earliest hour of change on eight dates would not be identified. If we use only the correlated resource use counters, then the earliest hour of change on one date would not be identified. Our results show that we require both the correlated resource use counters and correlated errors for identifying the earliest hour of change in the system behaviour on all dates. We found that the time-window between the hours of change on all the dates are different. The time-window ranges from one-hour to 15-hours.

Validation

Next, we test the significance of the correlation coefficient of: (i) groups of resource use counters that are strong positive correlated, and (ii) groups of error events that are strong positive correlated. We tested all the correlation coefficients against the null hypothesis. We obtained the z-scores for all the correlation coefficients and a summary is provided in Table 4.8.

Table 4.8: Summary of z-scores. n contains the number of hours in one day of logs.

Correlated groups	June 2011	July 2011	Aug 2011
Infiniband & Lustre I/O resource use counters ($n = 24$)	$3.71 \leq z_r \leq 10.68$	$3.58 \leq z_r \leq 10.68$	$6.51 \leq z_r \leq 10.68$
Communication & filesystem errors ($n = 24$)	$3.71 \leq z_e \leq 10.68$	$5.29 \leq z_e \leq 10.68$	$z_e = 10.68$

From Table 4.8, we observe that the z-scores for all the correlation coefficients range from 3.58 to 10.68. At the 99% confidence level, under the null hypothesis $z_{0r} = 2.64$ and $z_{0e} = 2.64$. Hence, we reject the null hypothesis in favour of the alternate hypothesis.

Next, for all hypothesis tests we use the significance level, $\alpha = 0.01$ and apply a one-sided test to obtain a P-value. We use the P-value for determining the probability of rejecting the null hypothesis when it is true. Table 4.8 summarises the z-scores for all the correlation coefficients. We observe that the smallest z-score is 3.58. Since this is a one-sided test, the P-value is equal to the probability of observing a value greater than 3.58 in the standard normal distribution, or $P(Z > 3.58) = 1 - P(Z \leq 3.58) = 1 - 0.999828 = 0.000172$. To account for the inflation in false positive due to multiple independent tests, we obtain the adjusted P-value $0.000172 \times 24 = 0.0041$ where 24 is the number of dates. The adjusted P-value is less than 0.01, indicating it is highly unlikely this result would be observed under the null hypothesis. From Table 4.8, we observe that all the z-scores are greater than or equal to 3.58. Therefore, the adjusted P-value for all the z-scores are less than 0.01, indicating it is highly unlikely these results would be observed under the null hypothesis.

4.4.3 Chipset and ECC Memory System

In this error case, we determine: (i) correlations between CPU and memory resource use counters, and (ii) correlations between chipset and ECC memory errors. We use the correlations to diagnose memory errors. Then, we assess the reliability of the ECC memory system.

Phase 1: Correlated CPU & Memory Resource Use Counters

When a user application is using the CPU, the resource use counter named `CPU user` is incremented. When a system application is using the CPU, the resource use counter named `CPU system` is incremented. When a memory page has not been accessed recently in the main memory, the resource use counter named `MEM Inactive` is incremented. When a memory page is recently accessed in the main memory, the

resource use counter named `MEM Active` is incremented. The CPU and memory resource use counters can be used to see what happens when CPU and memory activities occur.

Figure 4.20 shows the correlations between the resource use counters CPU user, memory active and memory inactive in June 2011.

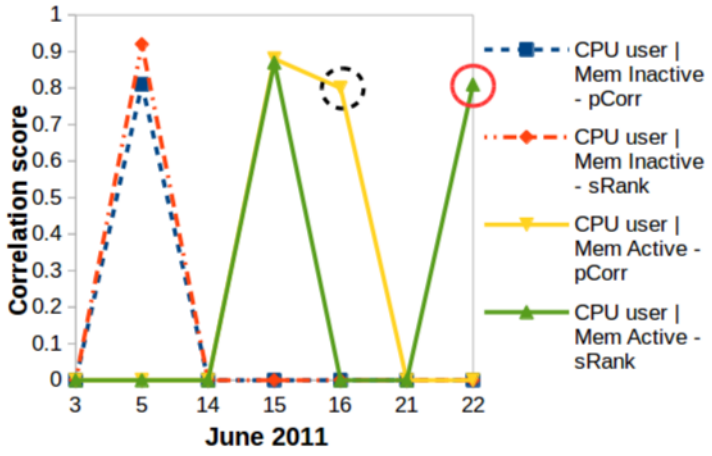


Figure 4.20: The full-circled counters were identified by Spearman-Rank correlation only. The dot-circled counters were identified by Pearson correlation only.

Figure 4.21 shows the correlations between the resource use counters CPU user, memory active and memory inactive in July 2011.

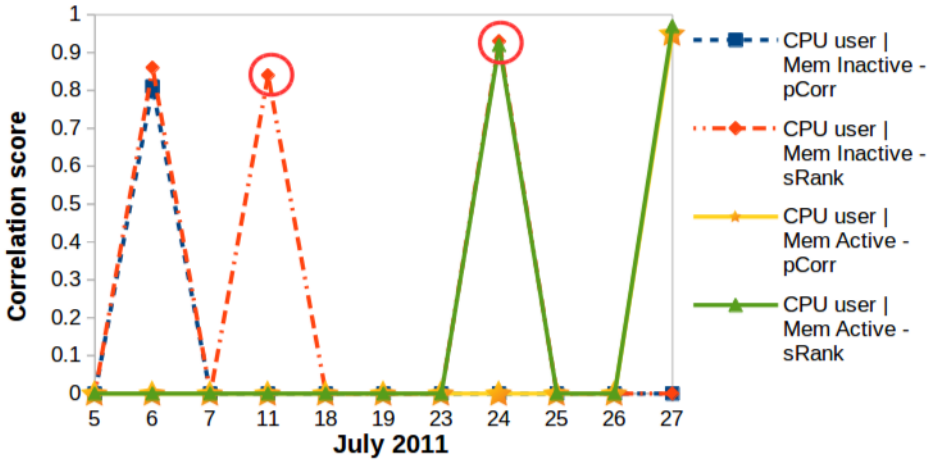


Figure 4.21: The full-circled counters were identified by Spearman-Rank correlation only.

Figure 4.22 shows the correlations between the resource use counters CPU user, memory active and memory inactive in August 2011.

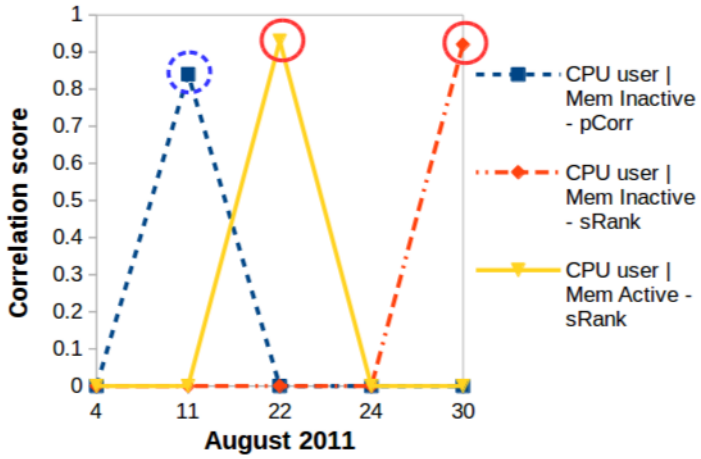


Figure 4.22: The full-circled counters were identified by Spearman-Rank correlation only. The dot-circled counters were identified by Pearson correlation only.

From Figure 4.20, Figure 4.21 and Figure 4.22, we observe that there is a strong positive correlation between CPU user and memory access. We identified: (i) correlations between `CPU user` and `MEM Inactive` with scores ranging from 0.81 to 0.93 on five dates, and (ii) correlations between `CPU user` and `MEM Active` with scores ranging from 0.80 to 0.97 on six dates. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one hour.

Figure 4.23 shows the correlations between the resource use counters CPU system, memory active and memory inactive in June 2011.

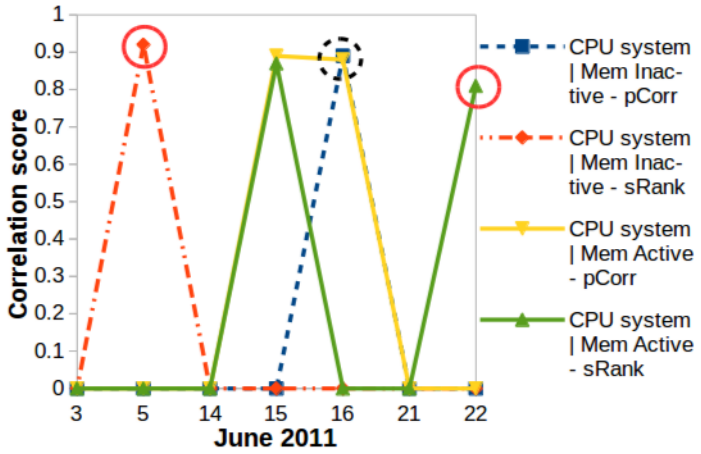


Figure 4.23: The full-circled counters were identified by Spearman-Rank correlation only. The dot-circled counters were identified by Pearson correlation only.

Figure 4.24 shows the correlations between the resource use counters CPU system, memory active and memory inactive in July 2011.

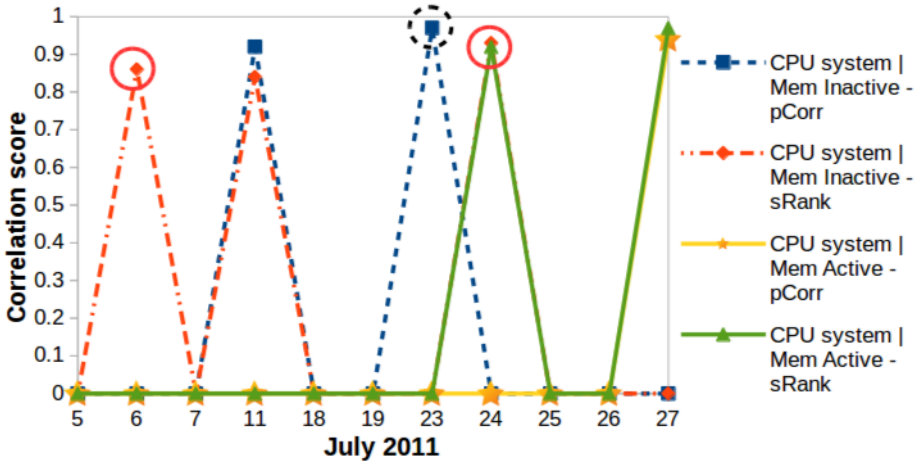


Figure 4.24: The full-circled counters were identified by Spearman-Rank correlation only. The dot-circled counters were identified by Pearson correlation only.

Figure 4.25 shows the correlations between the resource use counters CPU system, memory active and memory inactive in August 2011.

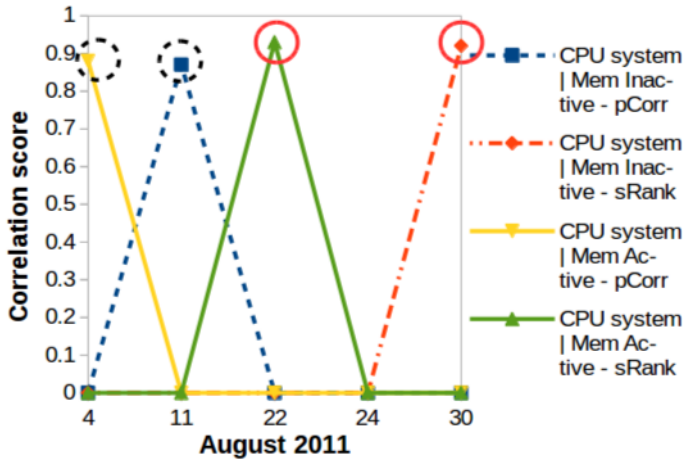


Figure 4.25: The full-circled counters were identified by Spearman-Rank correlation only. The dot-circled counters were identified by Pearson correlation only.

From Figure 4.23, Figure 4.24 and Figure 4.25, we observe that there is a strong positive correlation between CPU system and memory access. We identified: (i) correlations between `CPU system` and `MEM Inactive` with scores ranging from 0.84 to 0.97 on eight dates, and (ii) correlations between `CPU system` and `MEM Active` with scores ranging from 0.81 to 0.97 on seven dates. There are small changes in the

correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one hour.

We found that the correlated resource use counters were only identified by Pearson correlation on June 16, July 23, August 04 and 11. We found that the correlated resource use counters were only identified by Spearman-Rank correlation on seven different dates. If we use only Pearson correlation, then the correlated CPU and memory access resource use counters on June 05 and 22, July 06, 11 and 24, August 22 and 30 would not be identified. If we use only Spearman-Rank correlation, then the correlated CPU and memory access resource use counters on June 16, July 23, August 04 and 11 would not be identified. Our results show that:

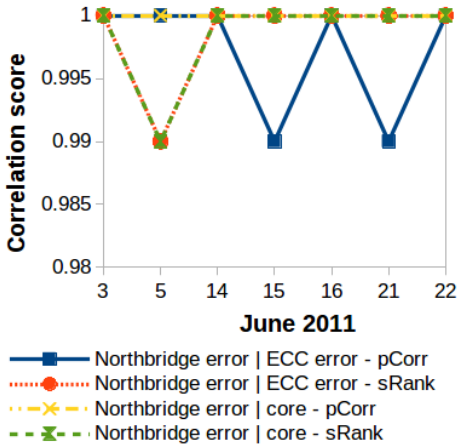
- There is a strong relationship of CPU user, CPU system, Memory Active and Memory Inactive resource use counters on 13 dates.
- Both Pearson and Spearman-Rank correlation methods are required. On four dates, the correlated CPU and memory access resource use counters were identified only by Pearson correlation. On seven more dates, the correlated CPU and memory access resource use counters were identified only by Spearman-Rank correlation.

In the second phase, we will use the correlations between two different groups of error events to diagnose ECC memory errors.

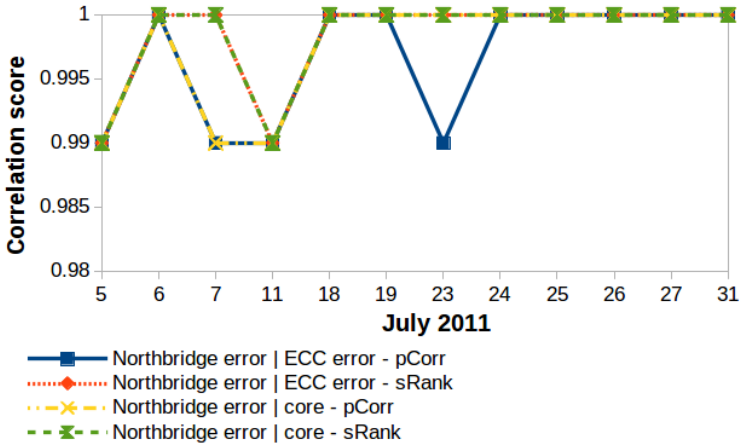
Phase 2: Correlated Chipset & ECC Memory Errors

The northbridge is a chip in the core logic chipset architecture on a computer motherboard. The CPU, memory and graphics controller communicate via the northbridge chip which is connected directly to the CPU. ECC memory is used in computers where internal data corruption can not be tolerated under any circumstances. When the CPU attempts to access corrupted data stored in ECC memory, the northbridge reports an error by generating a `Northbridge error` message, a CPU core `core` message that contains the CPU number that attempted to access the data and an ECC error `ECC error` message.

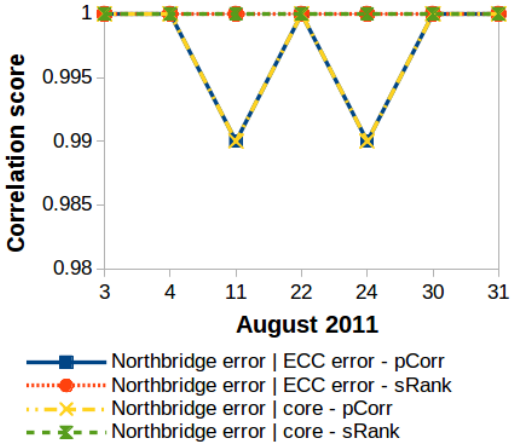
From Figure 4.26a, Figure 4.26b and Figure 4.26c, we observe that there is a strong positive correlation of northbridge, CPU core and ECC errors. We identified: (i) correlations between `Northbridge error` and `ECC error` events with scores ranging from 0.99 to 1 on 26 dates, and (ii) correlations between `Northbridge error` and `core` events with scores ranging from 0.99 to 1 on 26 dates. We observe that the dates of correlated northbridge error, core and ECC memory errors coincide with the dates of correlated CPU and memory access activity. There are small



(a) June 2011.



(b) July 2011.



(c) August 2011.

Figure 4.26: Correlations of northbridge error, core and ECC error events.

changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one hour.

We found that the correlated events were identified by both Pearson and Spearman-Rank correlation methods on all the 26 dates. When both Pearson and Spearman-Rank correlation methods identify the correlated events, Pearson correlation can be used as the primary method. Our results show that internal data corruption occurred daily on the Ranger HPC system.

Correlations with failures: Next, we determine the correlation strength of northbridge, CPU core and ECC memory errors to soft lockup events. We implemented a function that scans the list of correlated events for `Northbridge error`, `core`, `ECC error` and `soft lockup`. We found that the `Northbridge error`, `core` and `ECC error` events were weakly correlated to soft lockup events on all the 26 dates. This represents a recovery rate of 100%.

Combining analysis of CPU and memory access resource use counters with ECC memory errors provides the following benefit: When on the same day, CPU and memory access resource use counters are correlated and northbridge, CPU core and ECC error events are also correlated, it shows that CPU and memory access activities are associated with the generation of ECC memory errors. Therefore, we can use the correlated CPU and memory access resource use counters with the correlated CPU core, ECC and northbridge error events to monitor the state of the ECC memory recovery protocol.

Phase 3: Earliest Hour of Change

Table 4.9 shows the earliest hour of change in the correlated CPU and memory access resource use counters and correlated northbridge, ECC error and CPU core events. On all the dates, we observe that there are different earliest hour of change. On five dates, the earliest hour of change is associated with the correlated CPU and memory access resource use counters. On seven dates, the earliest hour of change is associated with the correlated chipset and ECC errors. On one date, the earliest hour of change is associated with both the correlated resource use counters and correlated errors . If we use only the correlated chipset and ECC error events, then the earliest hour of change on five dates would not be identified. If we use only the correlated CPU and memory access resource use counters, then the earliest hour of change on seven dates would not be identified. Our results show that we require both the correlated CPU and memory access resource use counters and correlated chipset and ECC error events for identifying the earliest hour of change in the system behaviour on all the dates. We found that there are different time-windows between the hour of change

Table 4.9: Hours associated with the correlated CPU and memory resource use counters and correlated chipset and ECC errors on Ranger.

Correlated counters	**Jun 5**	**Jun 15**	**Jun 16**	**Jun 22**	**Jul 6**	**Jul 11**	**Jul 23**	**Jul 24**	**Jul 27**
CPU & memory	7 PM	12 AM	6 AM	8 PM	2 PM	12 PM	4 AM	12 PM	10 PM
Correlated errors	**Jun 5**	**Jun 15**	**Jun 16**	**Jun 22**	**Jul 6**	**Jul 11**	**Jul 23**	**Jul 24**	**Jul 27**
Chipset & ECC errors	12 AM	1 PM	6 PM	2 AM	4 AM	12 PM	12 AM	2 PM	5 PM
Correlated counters	**Aug 4**	**Aug 11**	**Aug 22**	**Aug 30**					
CPU & memory	2 PM	12 PM	5 AM	1 PM					
Correlated errors	**Aug 4**	**Aug 11**	**Aug 22**	**Aug 30**					
Chipset & ECC errors	8 AM	2 PM	2 AM	6 PM					

identified on all the dates. The time-window ranges from one hour to 19 hours.

Validation

Next, we test the significance of the correlation coefficient of: (i) groups of resource use counters that are strong positive correlated, and (ii) groups of error events that are strong positive correlated. We tested all the correlation coefficients against the null hypothesis. We obtain the z-scores for all the correlation coefficients and a summary is provided in Table 4.10.

Table 4.10: Summary of z-scores. n contains the number of hours in one day of logs.

Correlated groups	**June 2011**	**July 2011**	**Aug 2011**
CPU & Memory counters ($n = 24$)	$3.74 \leq z_r \leq 5.86$	$3.74 \leq z_r \leq 8.16$	$4.17 \leq z_r \leq 6.18$
Chipset & ECC errors ($23 \leq n \leq 24$)	$z_e = 10.68$	$z_e = 10.68$	$z_e = 10.68$

From Table 4.10, we observe that the z-scores for all the correlation coefficients range from 3.74 to 10.68. At the 99% confidence level, under the null hypothesis $z_{0r} = 2.64$ and $z_{0e} = 2.64$. Hence, we reject the null hypothesis in favour of the alternate hypothesis.

Next, for all hypothesis tests we use the significance level, $\alpha = 0.01$ and apply a one-sided test to obtain a P-value. We use the P-value for determining the probability of rejecting the null hypothesis when it is true. Table 4.10 summarises

the z-scores for all the correlation coefficients. We observe that the smallest z-score is 3.74. Since this is a one-sided test, the P-value is equal to the probability of observing a value greater than 3.74 in the standard normal distribution, or $P(Z > 3.74) = 1 - P(Z \leq 3.74) = 1 - 0.99992 = 0.00008$. To account for the inflation in false positive due to multiple independent tests, we obtain the adjusted P-value $0.00008 \times 26 = 0.00208$ where 26 is the number of dates. The adjusted P-value is less than 0.01, indicating it is highly unlikely this result would be observed under the null hypothesis. From Table 4.10, we observe that all the z-scores are greater than or equal to 3.74. Therefore, the adjusted P-value for all the z-scores are less than 0.01, indicating it is highly unlikely these results would be observed under the null hypothesis.

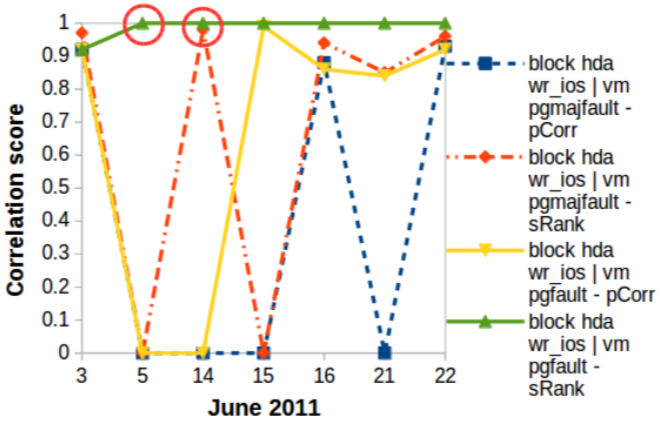
4.4.4 Linux Memory Management

In this error case, we determine: (i) the correlations between hard disk and virtual memory resource use counters, (ii) the correlations between file access and process errors, and (iii) the correlations between process errors and system memory exhaustion events. We use the correlations to diagnose Linux memory management problems. Then, we assess the reliability of Linux memory management.

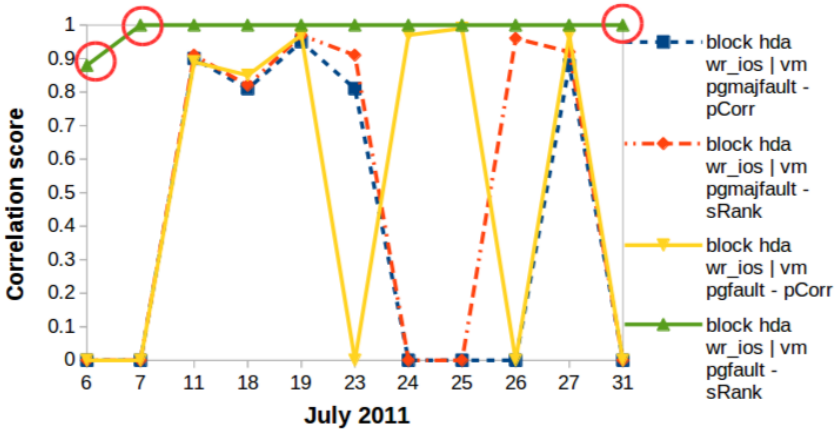
Phase 1: Correlated Harddisk and Virtual Memory Resource Use Counters

When a harddisk executes I/O write operations, the resource use counter named `block hda wr_ios` is incremented. When a number of sectors are written on the harddisk, the resource use counter named `block hda wr_sectors` is incremented. When a minor page fault occurs, the resource use counter named `vm pgfault` is incremented. When a major page fault occurs, the resource use counter named `vm pgmajfault` is incremented. The hard disk and virtual memory resource use counters can be used to see what happens when secondary storage (i.e., the hard disk) is used as memory by the Linux operating system.

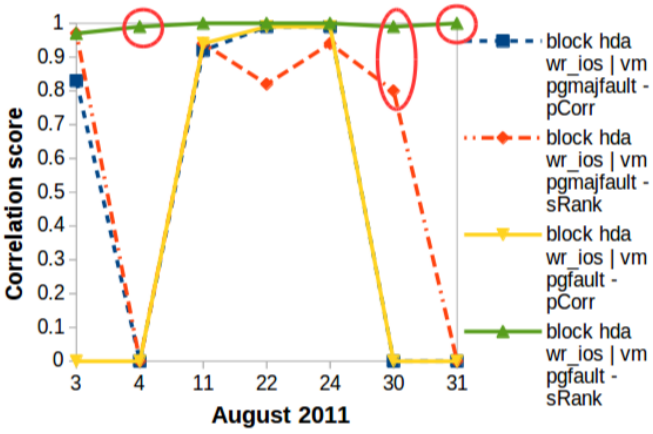
Figure 4.27a, Figure 4.27b and Figure 4.27c show the correlations of the resource use counters harddisk I/O write, minor and major page-fault in June, July and August 2011 respectively. From Figure 4.27a, Figure 4.27b and Figure 4.27c, we observe that there is a strong positive correlation between the resource use counters harddisk I/O write, minor and major page-fault. We identified: (i) correlations between `block hda wr_ios` and `vm pgmajfault` with scores ranging from 0.8 to 0.99 on 19 dates, and (ii) correlations between `vm pgfault` and `block hda wr_ios` with scores ranging from 0.8 to 1 on 25 dates. There are small changes in the



(a) Correlated harddisk I/O write, minor and major page-fault resource use counters in June 2011.



(b) Correlated harddisk I/O write, minor and major page-fault resource use counters in July 2011.



(c) Correlated harddisk I/O write, minor and major page-fault resource use counters in August 2011.

Figure 4.27: The full-circled counters were identified by Spearman-Rank correlation only.

correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1.

Figure 4.28 shows the correlations of the resource use counters that record harddisk sector write, minor page-fault and major page-fault in June 2011.

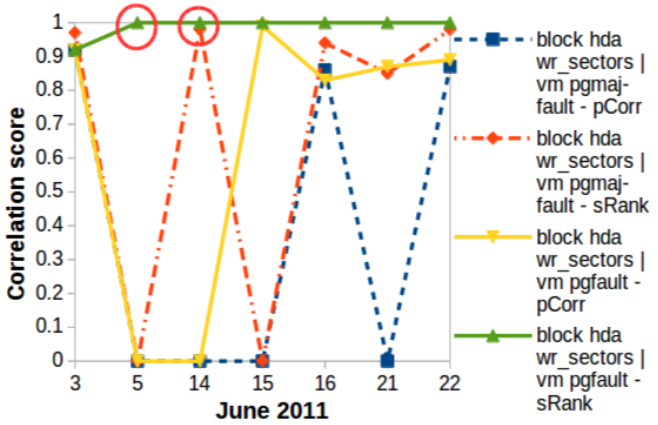


Figure 4.28: The full-circled counters were identified by Spearman-Rank correlation only.

Figure 4.29 shows the correlations of the resource use counters that record harddisk sector write, minor page-fault and major page-fault in July 2011.

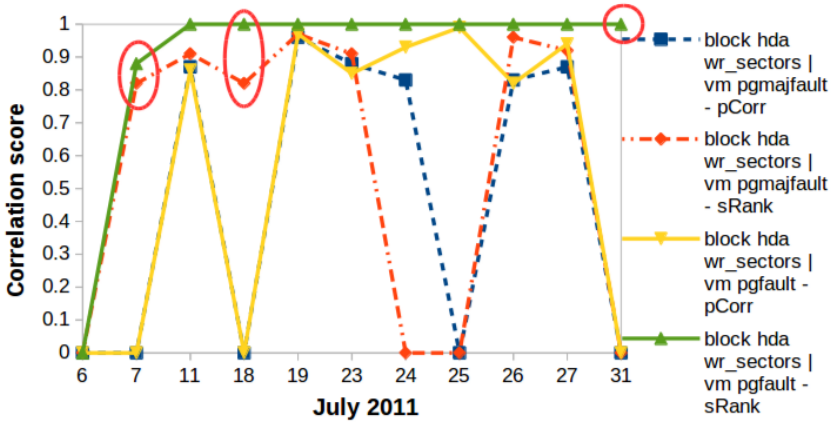


Figure 4.29: The full-circled counters were identified by Spearman-Rank correlation only.

From Figure 4.28, Figure 4.29 and Figure 4.30, we observe that there is a strong positive correlation between the resource use counters harddisk sector write, minor and major page-fault. We identified: (i) correlations between `block hdd wr_sectors` and `vm pgmajfault` with scores ranging from 0.8 to 0.99 on 20 dates, and (ii) correlations between `vm pgfault` and `block hdd wr_sectors`

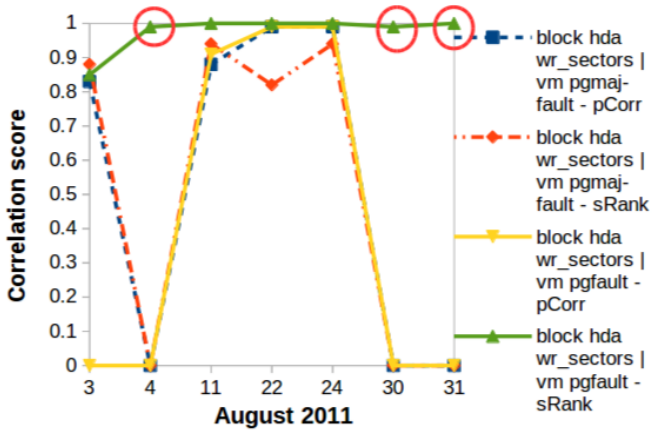


Figure 4.30: The full-circled counters were identified by Spearman-Rank correlation only.

with scores ranging from 0.8 to 1 on 25 dates. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one-hour.

We found that only the Spearman-Rank correlation method identified the correlated harddisk I/O write, harddisk sector write, minor and major page-fault resource use counters on all dates. If Spearman-Rank correlation alone identified the correlations on all the dates, then it can be used as the primary method. Our results show that:

- When the Linux operating system uses the hard disk as memory, there is a strong positive correlation between the resource use counters harddisk I/O write, harddisk sector write, minor page-fault and major page-fault.
- The Spearman-Rank correlation method can be used as the primary method to identify patterns of Linux system memory paging. It identified patterns of page-fault and harddisk I/O activities that follow a monotonically increasing function.

Phase 2: Correlated File Access and Process Errors

We identify file access errors by searching the system logs for messages that contain the keywords `read_lock_failed` and `write_lock_failed`. Information about a process state, name, memory address and running CPU is provided in a `Pid: comm` message. We implemented three functions to search and retrieve file access errors and process messages. When multiple processes access the same file, the filesystem

uses read and write locks to prevent other processes from writing to the same file. This is to ensure that the consistency of the file is maintained. In most cases, the process will complete its file I/O activity. It releases the lock and give it to the next process. If a process hanged while it is writing to the file, then the process may fail to release the lock. When another process attempts to access the same file, a deadlock can occur.

Figure 4.31 shows the correlations of the error messages read lock failed, write lock failed and process information in June 2011.

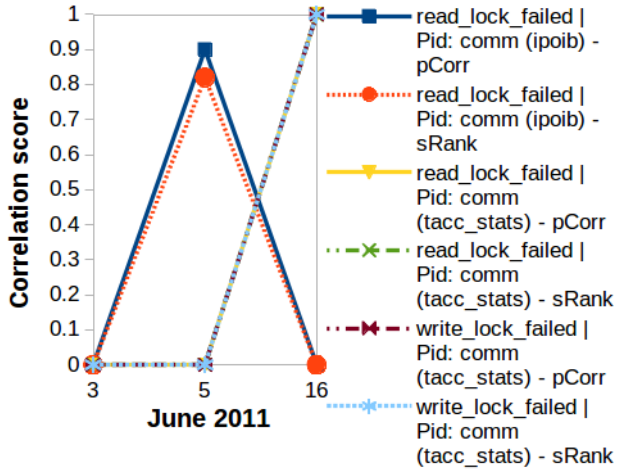


Figure 4.31: Correlation of messages read lock failed, write lock failed and Pid: comm.

Figure 4.32 shows the correlations of the error messages read lock failed, write lock failed and process information in July 2011.

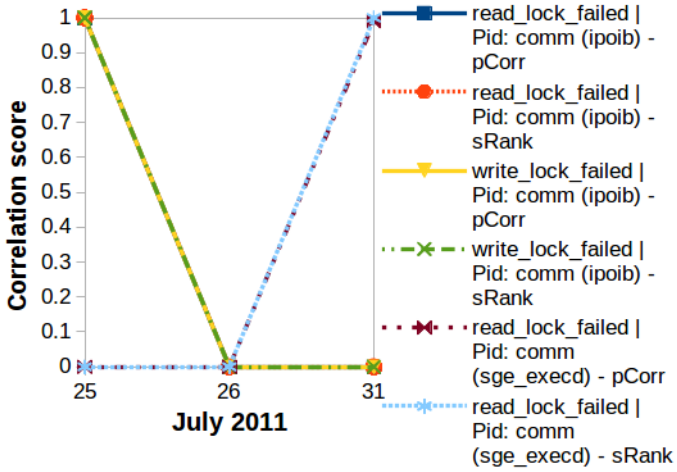


Figure 4.32: Correlation of messages read lock failed, write lock failed and Pid: comm.

Figure 4.33 shows the correlations of the error messages read lock failed, write lock failed and process information in August 2011.

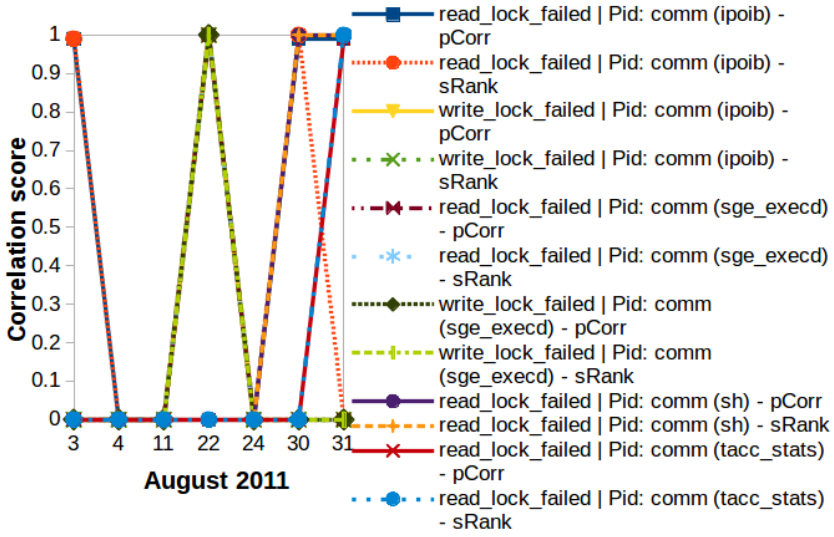


Figure 4.33: Correlation of messages read lock failed, write lock failed and Pid: comm.

From Figure 4.31, Figure 4.32 and Figure 4.33, we observe that there is a strong positive correlation between file access errors and process messages. We identified: (i) correlations between `read_lock_failed` and `Pid: comm` with scores ranging from 0.82 and 1 on eight dates, and (ii) correlations between `write_lock_failed` and `Pid: comm` with a score of 1 on three dates. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one hour.

We found that all the dates of the correlated file access errors and process messages coincide with the dates of correlated harddisk and virtual memory page-fault resource use counters. On the eight dates, we found that both Pearson and Spearman-Rank correlation methods identified: (i) the correlated `read_lock_failed` and `Pid: comm` events, and (ii) the correlated `write_lock_failed` and `Pid: comm` events. If both Pearson correlation and Spearman-Rank correlation methods identified the correlations on all the dates, then Pearson correlation can be used as the primary method. Our result shows that both Pearson and Spearman-Rank correlation identified the correlated file access and process errors on all the eight dates. Hence, Pearson correlation can be used as the primary method.

We manually scanned the `Pid: comm` and `soft lockup` messages to identify the name of the process. The name of the process in both `Pid: comm` and `soft lockup` messages are: `ipoib`, `tacc_stats`, `sge_execd` and `sh`. `ipoib` defines how IP packets

are sent over an Infiniband network. `tacc_stats` is a job-oriented and logically structured resource use monitor. `sge_execd` controls the job queues local to the machine and runs the jobs sent by the Sun Grid Engine master on the local queues. `sh` is a Linux operating system command-line interpreter.

Correlations with failures: We determine: (i) the process name in the soft lockup message that is strongly positive correlated to the name in the process message, and (ii) the dates read and write lock failed events are strongly positive correlated to the soft lockup events. We manually scanned the list of correlated events to identify: (i) the correlation score between the process and soft lockup messages, and (ii) the correlation score between the read and write lock failed events and soft lockup message. We summarise the correlations in Tables 4.11 and 4.12.

Table 4.11: Summary of names of soft lockup processes and dates of correlated "Pid: comm" and soft lockup on Ranger.

Process error	**soft lockup**	**Date**	**pCorr**	**sRank**
Pid: comm (ipoib)	ipoib	June 5	0.99	0.97
		July 25	0.99	nil
		Aug 3	1	0.99
		Aug 22	1	1
		Aug 30	1	1
		Aug 31	1	1
Pid: comm (tacc_stats)	tacc_stats	June 16	1	1
		Aug 31	1	1
Pid: comm (sge_execd)	sge_execd	July 31	1	1
		Aug 22	1	1
Pid: comm (sh)	sh	Aug 30	1	1
		Aug 31	1	1

From Table 4.11, we observe that the process name in all the soft lockup messages matched the name of the process in all the process messages. Further, we observe that all the soft lockup message dates matched all the dates of correlated file access errors and process messages. We found that only the Pearson correlation method identified these correlations on all the dates. Hence, Pearson correlation can be used as the primary method.

From Table 4.12, we observe that the dates of soft lockups matched all the dates of correlated file access errors and process messages. We found that only the Pearson correlation method identified these correlations on all the dates. Hence, Pearson correlation can be used as the primary method.

Detailed diagnosis: When a process attempt to read or write data to the filesystem but the read or write locks have not been released, the process generates a "Pid: comm" error message. On a total of eight dates in June, July and August 2011,

Table 4.12: Dates of correlated file access and soft lockup on Ranger.

File-access error	soft lockup	Date	pCorr	sRank
read_lock_failed	ipoib	June 5	0.85	0.87
		July 25	0.99	nil
		Aug 3	0.99	0.99
		Aug 22	1	1
		Aug 30	0.99	1
		Aug 31	0.99	nil
write_lock_failed	ipoib	Aug 22	1	1
read_lock_failed	tacc_stats	June 16	1	1
		Aug 31	0.99	1
write_lock_failed	tacc_stats	June 16	1	1
read_lock_failed	sge_execd	July 31	0.99	1
		Aug 22	1	1
write_lock_failed	sge_execd	Aug 22	1	1
read_lock_failed	sh	Aug 30	1	1
		Aug 31	1	1

the processes failed to recover from the filesystem read and write lock failures. Our result shows that when the filesystem read or write lock is not released, the process was unable to access data on the filesystem which led to compute node soft lockup on eight out of eight dates. This represents a failure rate of 100%.

The benefit of combining analyses of hard disk and virtual memory resource use counters with file access and process errors is given as follows: When correlations of hard disk I/O and virtual memory page-faults and correlations of file access and process errors occur on the same day, it shows that the Linux operating system memory paging activities are associated with the generation of file access errors and process messages. Therefore, we can use the correlations to monitor the state of Linux memory usage.

Phase 2: Correlated Process Errors and System Memory Exhaustion

We identify system memory exhaustion by searching the system logs for keywords `system memory exhausted`. When a process is created, the operating system allocates memory to the process. In most cases, memory is allocated to the process successfully. However, in a rare case another process can be created when there is no more memory available. The operating system is no longer able to allocate memory for the new process. The operating system starts to move data from memory to harddisk to make memory available for the new process.

From Table 4.13, we observe that there is a strong positive correlation between the process message and system memory exhausted events. We identified correlations

Table 4.13: Correlation of "Pid: comm" and "system memory exhausted" events on Ranger.

Pid: comm event	Error event	Date	pCorr	sRank
bash	system memory exhausted	June 5	0.99	-
sshd	system memory exhausted	June 21	1	1

between `Pid: comm` and `system memory exhausted` events with scores ranging from 0.99 to 1 on two dates. The correlated process and system memory exhausted events coincide with two dates of correlated harddisk and virtual memory page-fault resource use counters. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one hour. We found that only the Pearson correlation method identified the correlated process message and system memory exhausted event on the two dates. If Pearson correlation alone identified the correlations on all the dates, then it can be used as the primary method. Our result shows that Pearson correlation identified the correlated process message and system memory exhausted event on the two dates. Hence, Pearson correlation can be used as the primary method.

We manually scanned the `Pid: comm` messages and identified the process names. They are: `sshd` and `bash`. `sshd` provides secure encrypted communication between two clients in an unsecured network. `bash` is the default command-line interpreter in the Linux operating system.

Correlations with failures: Next, we manually scanned the list of correlated events to determine the correlation strength between: (i) the `Pid: comm` and soft lockup events, and (ii) `system memory exhausted` and soft lockup events. A summary is given in Table 4.14.

Table 4.14: Summary of correlated "Pid: comm", "system memory exhausted" and soft lockup events on Ranger.

Error event	soft lockup	Date	pCorr	sRank
Pid: comm (bash)	bash	June 5	0.99	-
system memory exhausted	bash	June 5	1	-
Pid: comm	sshd	June 21	0.99	1
system memory exhausted	sshd	June 21	1	1

From Table 4.14, we observe that: (i) the processes bash and sshd are strongly correlated to soft lockup events with scores that range between 0.99 to 1, and (ii) the system memory exhausted events are strongly correlated to soft lockup events with scores that range between 0.99 to 1. We found that only the Pearson correlation method identified correlations of process and system memory exhausted events with

soft lockup events on the two dates. Our result shows that Pearson correlation can be used as the primary method.

Detailed diagnosis: When the Linux operating system is unable to allocate memory for a process, a system memory exhausted message is generated. We showed that process error messages and system memory exhaustion messages are strongly positive correlated to soft lockup messages. Our result shows that when a process requested for memory on a system exhausted of memory, the system was unable to allocate memory to the process which led to compute node soft lockup on two out of two dates. This represents a failure rate of 100%.

The benefit of combining analyses of hard disk I/O and virtual memory resource use counters with process errors and system memory exhaustion messages is given as follows: When correlations of hard disk I/O and virtual memory resource use counters and correlations of process errors and system memory exhaustion messages occur on the same day, it shows that Linux O/S memory paging activities are associated with the generation of process errors and system memory exhaustion messages. Therefore, we can use the correlations to monitor the state of Linux memory usage.

Phase 3: Earliest Hour of Change

Table 4.15 shows the earliest hour of change in the correlated harddisk write and page fault resource use counters and the correlated file access errors and process messages. On all the dates, we observe that the earliest hour of change is different. On four dates, the earliest hour of change is associated with the correlated harddisk write and page fault resource use counters. On three dates, the earliest hour of change is associated with the correlated file access errors and process messages. If we use only the correlated harddisk write and page fault resource use counters, then the earliest hour of change in system behaviour on three dates would not be identified. If we use only the correlated file access errors and process messages, then the earliest hour of change in system behaviour on four dates would not be identified.

Table 4.16 shows the earliest hour of change in the correlated harddisk write and page fault resource use counters and the correlated process error and system memory exhausted events.

From Table 4.16, we observe that the earliest hour of change is associated with the correlated errors on June 5. The earliest hour of change is associated with the correlated resource use counters on June 21. Our results show that we require both the correlated resource use counters and correlated errors to identify the earliest hour of change on eight out of nine dates. We found that there are different time windows between the hour of change on seven dates. The time-window ranges from

Table 4.15: Hours associated with the correlated HDD and virtual memory resource use counters and correlated file access and process errors.

Ranger				
Correlated counters	**June 5**	**June 16**	**July 25**	**July 31**
HDD I/O & virtual memory	1 AM	12 AM	12 PM	6 AM
Correlated errors	**June 5**	**June 16**	**July 25**	**July 31**
File-access & process errors	1 AM	3 PM	3 PM	4 AM
Correlated counters	**Aug 3**	**Aug 22**	**Aug 30**	**Aug 31**
HDD I/O & virtual memory	12 PM	10 PM	12 PM	12 PM
Correlated errors	**Aug 3**	**Aug 22**	**Aug 30**	**Aug 31**
File-access & process errors	5 AM	9 AM	7 PM	4 PM

Table 4.16: Hours associated with the correlated HDD and virtual memory resource use counters and correlated process errors and system memory exhausted events.

Ranger		
Correlated counters	**June 5**	**June 21**
HDD I/O & virtual memory	1 AM	9 PM
Correlated errors	**June 5**	**June 21**
Process errors & system memory exhausted	12 AM	10 PM

one hour to 15 hours.

Validation

Next, we test the significance of the correlation coefficient of: (i) groups of resource use counters that are strong positive correlated, and (ii) groups of error events that are strong positive correlated. We tested all the correlation coefficients against the null hypothesis. We obtained the z-scores for all the correlation coefficients and a summary is given in Table 4.17. We observe that the z-scores for all the correlation coefficients range from 5.31 to 12.13. At the 99% confidence level, under the null hypothesis $z_{0r} = 2.64$ and $z_{0e} = 2.64$. Hence, we reject the null hypothesis in favour of the alternate hypothesis.

Next, for all hypothesis tests we use the significance level, $\alpha = 0.01$ and apply a one-sided test to obtain a P-value. We use the P-value to determine the

Table 4.17: Summary of z-scores. n_r contains the number of hours in one day of resource usage logs. n_e contains the number of hours in one day of system logs.

Ranger			
Correlated counters	**June 2011**	**July 2011**	**Aug. 2011**
HDD I/O & virtual memory ($n_r = 24$)	$5.31 \leq z_r \leq 12.13$	$5.31 \leq z_r \leq 12.13$	$5.31 \leq z_r \leq 12.13$
Correlated errors	**June 2011**	**July 2011**	**Aug. 2011**
File access & process errors ($17 \leq n_e \leq 24$)	$6.72 \leq z_e \leq 12.13$	$z_e = 12.13$	$z_e = 12.13$
Process errors & system memory exhaustion ($n_e = 24$)	$z_e = 12.13$	-	-

probability of rejecting the null hypothesis when it is true. Table 4.17 summarises the z-scores for all the correlation coefficients. We observe that the smallest z-score is 5.31. Since this is a one-sided test, the P-value is equal to the probability of observing a value greater than 5.31 in the standard normal distribution, or $P(Z > 5.31) = 1 - P(Z \leq 5.31) = 1 - 0.99999 = 0.00001$. To account for inflation in false positive due to multiple independent tests, we obtain the adjusted P-value $0.00001 \times 26 = 0.00026$, where 26 is the total number of days of Ranger's log-data. The P-value is less than 0.01, indicating it is highly unlikely this result would be observed under the null hypothesis. From Table 4.17, we observe that all the z-scores are greater than or equal to 5.31. Therefore, the adjusted P-value for all the z-scores are less than 0.01, indicating it is highly unlikely these results would be observed under the null hypothesis.

4.5 Summary

In this chapter, we presented the CORRMEXT framework that correlated both the resource use data and system logs to identify: (i) frequently occurring error cases, (ii) report the success and failure of error recovery protocols. The main technical contribution is a new systems diagnostics framework that integrated data type extraction, multiple correlation methods and time-bin variance extraction. We applied CORRMEXT on the TACC_Stats resource use data and Rationalized message logs on the Ranger HPC system. CORRMEXT diagnosed five error cases and extracted the variance in the times of the correlated resource use counter groups and correlated error groups to identify the earliest occurrences of the problem. To ensure diagnostics accuracy, CORRMEXT used the Bonferroni correction and showed that all the correlations are significant. We showed that CORRMEXT can identify the error cases that occurred frequently and report the success and failure of error

recovery protocols.

Here, we provide our recommendation on what one should look for in the resource use data and system logs. In the resource use data, we can use the following types of resource use counters to do the following: (i) amount of NUMA miss, amount of NUMA foreign, number of Linux processes created and number of context switching to monitor memory allocation when a new Linux process is created, (ii) number of network packets dropped, number of network packets transmitted, number of bytes read on the filesystem and number of bytes written to the filesystem to monitor usage of the network and filesystem, (iii) amount of CPU resources consumed by a user application, amount of CPU resources consumed by a system application, amount of active memory pages and amount of inactive memory pages to monitor CPU consumption and memory access, (iv) number of harddisk I/O writes, number of harddisk sectors written, number of minor page-faults and number of major page-faults to monitor Linux memory management.

In the system logs, we can use the following types of messages to do the following: (i) segmentation fault and general protection error messages to identify a memory leak caused by a faulty program, (ii) error communicating with, failure inode and directory read error messages to identify a cause of Lustre filesystem and Lustre client communication error, (iii) northbridge error, ECC error and core messages to identify occurrences of data corruption in memory, (iv) read lock failed, write lock failed and information about a system process to identify a hung process caused by a lock on the filesystem, (v) system memory exhausted and information about a system process to identify a hung process caused by system memory exhaustion.

Chapter 5

Generalising CORRMEXT on Multiple HPC Systems

In this chapter, we apply the CORRMEXT framework described in Chapter 4 on multiple HPC systems. We combine analysis of system logs with resource utilisation data and present several new findings and failure patterns. There are no system logs available on Stampede-1. Therefore, on Stampede-1 we focus on the resource use data. We identified one error case and one resource usage activity case on Stampede-1. We identified two error cases on Lonestar4.

We structure the remainder of this chapter as follows: In Section 5.1, we describe the cluster log-data and error cases on the Lonestar4 and Stampede-1 HPC systems. In Section 5.2, we present the analysis for the error and activity cases on Stampede-1. one In Section 5.3, we present the analyses for two error cases on Lonestar4. In Section 5.4, we conclude with a summary and a recommendation.

5.1 Case Study Systems: Lonestar4 and Stampede-1

We conduct studies of frequently occurring error cases on the Lonestar4 and Stampede-1 HPC systems. The Lonestar4 HPC system was a 1,888 node Linux-based cluster. The Stampede-1 HPC system was a 6,400 nodes Linux-based cluster. The Lonestar4 and Stampede-1 HPC systems were operated by the Texas Advanced Computing Center at The University of Texas at Austin. We collected 26 days worth of resource use data and system logs on Lonestar4. We collected 28 days worth of resource use data on Stampede-1. The TACC_Stats resource use data was sampled at intervals of 10 minutes on Lonestar4 and Stampede-1. The dates of log-data analysed are given in Table 5.1.

When the Linux operating system kernel goes into a loop, i.e., Linux hanged,

Table 5.1: Dates of cluster log-data analysed.

Lonestar4	
Month	**Dates (26 days)**
February 2013	3 to 28
Stampede-1	
Month	**Dates (28 days)**
February 2017	1 to 28

a soft lockup event is generated. To identify the soft lockup event, we scan the system logs for a message containing the keywords `soft lockup`. We implemented a function in CORRMEXT to search the system logs for soft lockup events and extract the dates of soft lockups. In the reference [36], it is reported that there is a lead time of six hours from the occurrence of an error to a soft lockup event. We identified three dates of soft lockup events in the Lonestar4 syslogs. There are no system logs available on Stampede-1. The system logs generated on Lonestar4 contain messages generated from the Linux operating system kernel, Lustre filesystem and Linux processes.

We obtain the diagnostics reports generated by CORRMEXT. The diagnostics reports contain the lists of correlated resource use counters and correlated events. All the error cases are new ones and have not heretofore been reported in Chapter 4. The error cases are: (i) network data errors, (ii) network data and software errors, and (iii) filesystem, process and software errors. A list of the error cases is given in Table 5.2.

Table 5.2: List of error cases identified on Lonestar4 and Stampede-1.

System	**Component**	**Error**	**No. of dates**
Stampede-1	Infiniband network	Network data	27
Stampede-1	Storage system & Linux processes	–	27
Lonestar4	Infiniband network	Network data & software	6
Lonestar4	Storage system & Linux processes	Filesystem, process & software	6

5.2 Stampede-1 HPC System

5.2.1 Network Data Errors

In this error case, we determine the correlations between the Infiniband network and compute node network interface resource use counters. We use the correlations to identify network data errors. Then, we assess the system reliability.

Phase 1: Correlated Infiniband and Compute Node Network Interface Resource Use Counters

When a compute node receives data on the network, TACC_Stats increments the resource use counter named `net eth0 rx_bytes` and the resource use counter named `net eth0 rx_packets`. "eth0" is the identifier for a compute node network interface card. When the Infiniband switch receives data on the network, the resource use counter named `net ib0 rx_bytes` and the resource use counter named `net ib0 rx_packets` are incremented. "ib0" is the identifier for the Infiniband switch. In most cases, both the compute node and Infiniband switch receive the data correctly. If the switch or a network card is faulty or wrongly configured, then it may not receive the data correctly. When the Infiniband switch receives corrupted data, the resource use counter named `net ib0 rx_frame_errors` and the resource use counter named `net ib0 rx_crc_errors` are incremented. When a compute node receives corrupted data, the resource use counter named `net eth0 rx_frame_errors` and the resource use counter named `net eth0 rx_crc_errors` are incremented. The Infiniband and compute node network interface card resource use counters can be used to see what happens when data errors occur on the network.

Figure 5.1 shows the correlation of data frame errors and network data received by the Infiniband switch and compute nodes on Stampede-1.

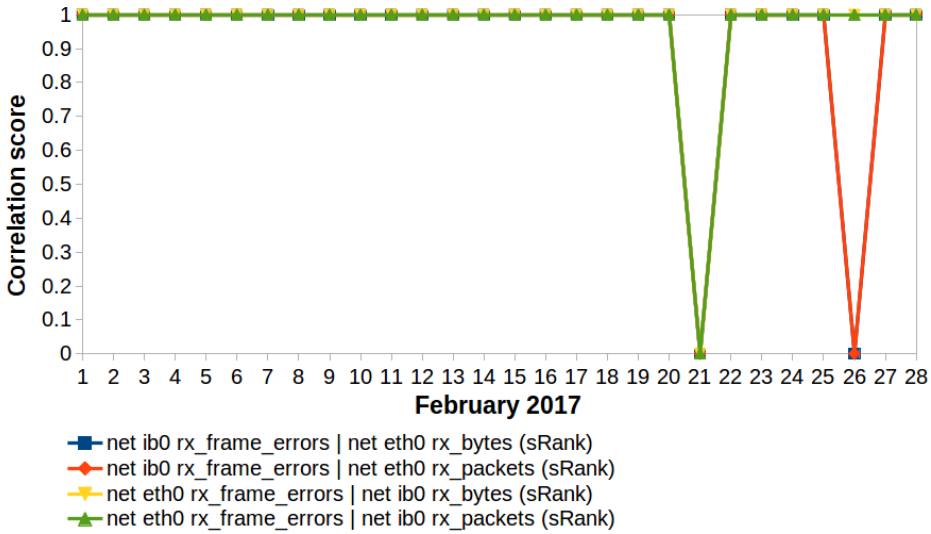


Figure 5.1: Correlation of data frame errors.

We observe that there is a strong positive correlation between the resource use counters that record network data frame errors and network data received. We identified: (i) correlations of `net ib0 rx_frame_errors` to `net eth0 rx_bytes` and `net eth0 rx_packets` with a score of 1 on 26 dates in February 2017, and (ii) correlations

of `net eth0 rx_frame _errors` to `net ib0 rx_bytes` and `net ib0 rx_packets` with a score of 1 on 27 dates in February 2017. We found that only the Spearman-Rank correlation method identified the correlated resource use counters on all 27 dates. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one-hour.

Figure 5.2 shows the correlation of cyclic redundancy check errors and network data received by the Infiniband switch and compute nodes on Stampede-1.

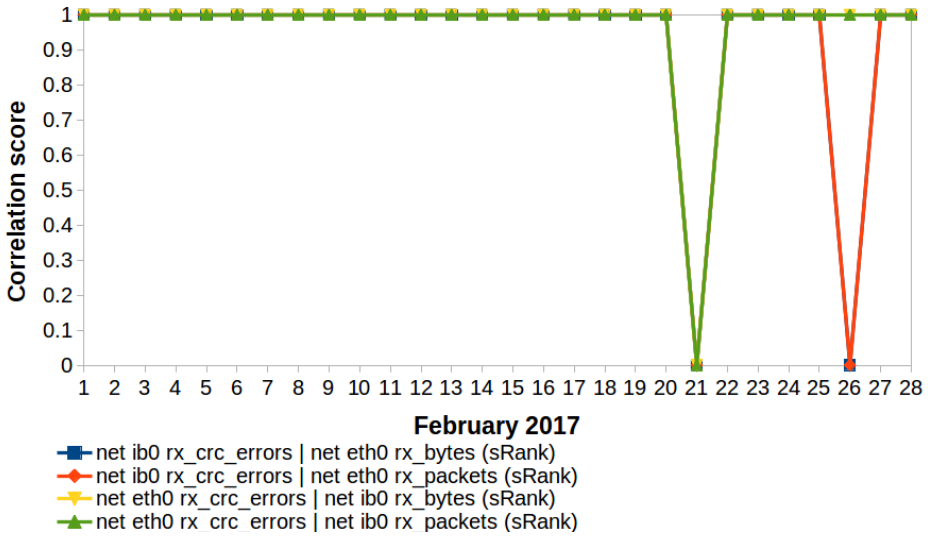


Figure 5.2: Correlation of cyclic redundancy check errors.

We observe that there is a strong positive correlation between the resource use counters that record network data CRC errors and network data received. We identified: (i) correlations of `net ib0 rx_crc_errors` to `net eth0 rx_bytes` and `net eth0 rx_packets` with a score of 1 on 26 dates in February 2017, and (ii) correlations of `net eth0 rx_crc_errors` to `net ib0 rx_bytes` and `net ib0 rx_packets` with a score of 1 on 27 dates in February 2017. We found that only the Spearman-Rank correlation method identified the correlated resource use counters on all 27 dates. If Spearman-Rank correlation alone identified the correlations on all the dates, then it can be used as the primary method. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one-hour. Our results show that:

- The Infiniband switch received data frame errors and CRC errors on 26 out of 28 days in February 2017, representing 92% of the dates.

- The compute nodes received data frame errors and CRC errors on 27 out of 28 days in February 2017, representing 96% of the dates.
- The Spearman-Rank correlation method can be used as the primary method to identify network data errors. It identified patterns of network data errors that follow a monotonically increasing function.

Validation

Next, we test the significance of the correlation coefficients of groups of resource use counters that are strong positive correlated. We tested all the correlation coefficients against the null hypothesis. We obtain the z-scores for all the correlation coefficients and a summary is provided in Table 5.3.

Table 5.3: Summary of z-scores. n_r contains the number of hours in one day of resource usage logs.

Stampede-1	
Correlated counters	**Feb 1 to 28, 2017**
Infiniband & compute node ($n_r = 24$)	$z_r = 12.13$

From Table 5.3, we observe that the z-scores for all the correlation coefficients are 12.13. At the 99% confidence level, under the null hypothesis $z_{0r} = 2.64$ and $z_{0e} = 2.64$. Hence, we reject the null hypothesis in favour of the alternate hypothesis.

Next, for all hypothesis tests we use the significance level, $\alpha = 0.01$ and apply a one-sided test to obtain a P-value. We use the P-value to determine the probability of rejecting the null hypothesis when it is true. Table 5.3 summarises the z-scores for all the correlation coefficients. We observe that all the z-scores is 12.13. Since this is a one-sided test, the P-value is equal to the probability of observing a value greater than 12.13 in the standard normal distribution, or $P(Z > 12.13) = 1 - P(Z \leq 12.13) = 1 - 0.99999 = 0.00001$. To account for inflation in false positive due to multiple independent tests, we obtain the adjusted P-value $0.00001 \times 28 = 0.00028$ where 28 is the number of days. All the z-scores are equal to 12.13. Therefore, the adjusted P-value for all the z-scores are less than 0.01, indicating it is highly unlikely these results would be observed under the null hypothesis.

5.2.2 Storage System and Linux Process

In this activity case, we determine the correlations of harddisk, filesystem and Linux processes resource use counters. We use the correlations to identify activities between the storage system and Linux operating system processes.

Phase 1: Correlated Harddisk, Filesystem and Linux Process Resource Use Counters

When a Linux process sends data to be read or written to the harddisk, three steps are executed to perform the operation. In the first step, the filesystem allocates an inode. TACC_Stats increments the resource use counter named `llite /work alloc_inode` which records the number of inodes allocated on the work partition of the filesystem. In the second step, it performs a seek operation. TACC_Stats increments the resource use counter named `llite /work seek` which records the number of seek operations on the work partition on the filesystem. In the third step, the data is written to the harddisk. TACC_Stats increments the resource use counter named `md0 wr_sectors` which records the number of sectors data is written to the harddisk. The resource use counter named `ps processes` records the number of Linux process created. The resource use counter named `ps ctxt` records the number of context switches across all the CPUs. The harddisk, filesystem and Linux process resource use counters can be used to see what happens when Linux processes write to the storage system.

Figure 5.3 shows the correlation of Linux process and filesystem resource use counters.

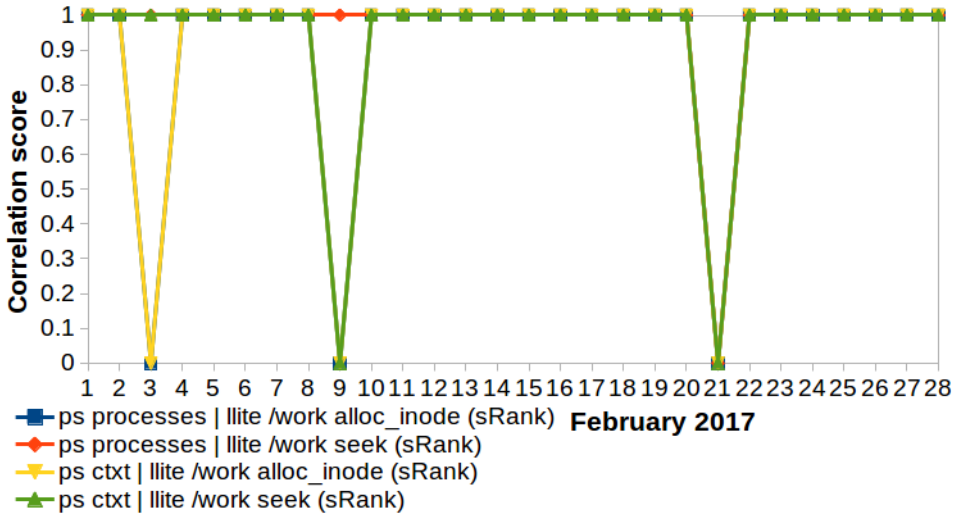


Figure 5.3: Correlation of Linux process & filesystem resource use counters.

From Figure 5.3 and Figure 5.4, we observe: (i) strong positive correlations of `md0 wr_sectors` to `ps processes` and `ps ctxt` with a score of 1 on 27 dates, and (ii) strong positive correlations of `ps processes` and `ps ctxt` to `llite /work alloc_inode` and `llite /work seek` with a score of 1 on 27 dates. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the

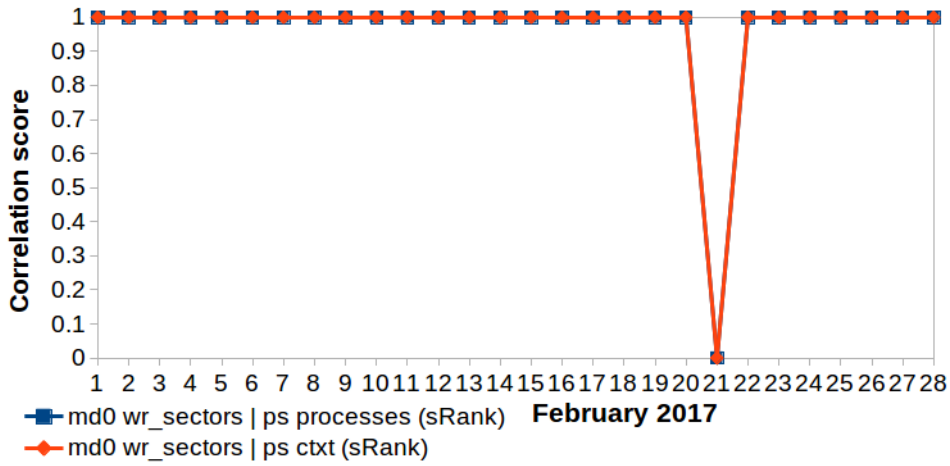


Figure 5.4: Correlation of Linux process & harddisk sector write resource use counters.

correlations obtained on time-bins of one hour. We found that only the Spearman-Rank correlation method identified the correlated resource use counters on all the dates. Our results show that:

- When a Linux process writes to the storage system, a strong correlation of harddisk and Lustre filesystem activities is generated.
- The Spearman-Rank correlation method can be used to identify activities on the harddisk and Lustre filesystem. It identified patterns of harddisk, filesystem and Linux process that follow a monotonically increasing function.

Validation

Next, we test the significance of the correlation coefficient of groups of resource use counters that are strong positive correlated. We tested all the correlation coefficients against the null hypothesis. We obtained the z-scores for all the correlation coefficients and a summary is given in Table 5.4.

Table 5.4: Summary of z-scores. n_r contains the number of hours in one day of resource use logs.

Stampede-1	
Correlated counters	**Feb 1 to 28, 2017**
HDD, filesystem & Linux process ($n_r = 24$)	$z_r = 12.13$

From Table 5.4, we observe that the z-scores for all the correlation coefficients is 12.13. At the 99% confidence level, under the null hypothesis $z_{0r} = 2.64$ and $z_{0e} = 2.64$. Hence, we reject the null hypothesis in favour of the alternate hypothesis.

Next, for all hypothesis tests we use the significance level, $\alpha = 0.01$ and apply a one-sided test to obtain a P-value. We use the P-value to determine the probability of rejecting the null hypothesis when it is true. We observe that all the z-scores is 12.13. The P-value is $P(Z > 12.13) = 1 - P(Z \leq 12.13) = 1 - 0.99999 = 0.00001$. To account for inflation in false positive due to multiple independent tests, we obtain the adjusted P-value $0.00001 \times 28 = 0.00028$, where 28 is the number of days. All the z-scores are equal to 12.13. Therefore, the adjusted P-value for all the z-scores are less than 0.01, indicating it is highly unlikely these results would be observed under the null hypothesis.

5.3 Lonestar4 HPC System

5.3.1 Network Data Errors and Network Software Errors

In this error case, we determine the correlations between: (i) the Infiniband network and compute node network interface resource use counters, and (ii) the correlations between DNS lookup failure and GSIFTP software messages. We use the correlations to diagnose network problems. Then, we assess the system reliability.

Phase 1: Correlated Infiniband and Compute Node Network Interface Resource Use Counters

When a compute node receives data on the network, TACC_Stats increments the resource use counter named `net eth0 rx_bytes` and the resource use counter named `net eth0 rx_packets`. "eth0" is the identifier for a compute node network interface card. When the Infiniband switch receives data on the network, the resource use counter named `net ib0 rx_bytes` and the resource use counter named `net ib0 rx_packets` are incremented. "ib0" is the identifier for the Infiniband switch. In most cases, both the compute node and Infiniband switch receive the data correctly. If the switch or a network card is faulty or wrongly configured, then it may not receive the data correctly. When the Infiniband switch receives corrupted data, the resource use counter named `net ib0 rx_frame_errors` and the resource use counter named `net ib0 rx_crc_errors` are incremented. When a compute node receives corrupted data, the resource use counter named `net eth0 rx_frame_errors` and the resource use counter named `net eth0 rx_crc_errors` are incremented. The Infiniband and compute node network interface card resource use counters can be used to see what happens when data errors occur on the network.

Figure 5.5 shows the correlation of resource use counters that record data frame errors and network data received by the Infiniband switch and compute nodes.

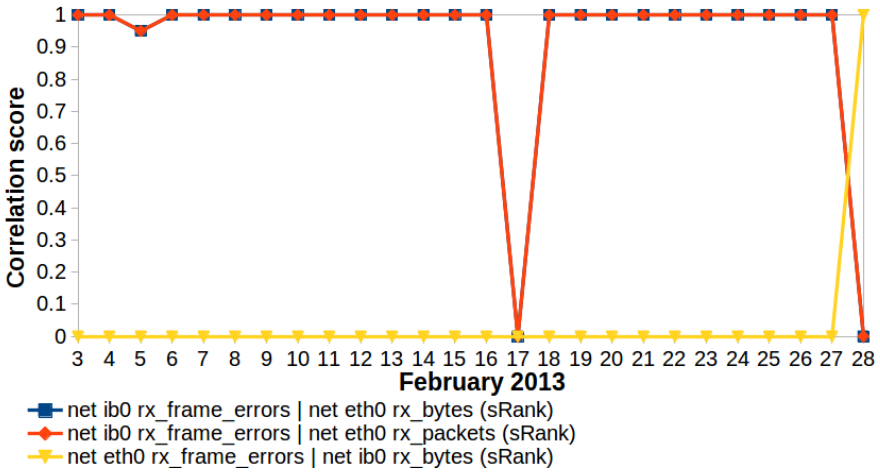


Figure 5.5: Correlation of data frame errors.

We observe that there is a strong positive correlation between the resource use counters data frame errors and network data received. We identified: (i) correlations of `net ib0 rx_frame_errors` to `net eth0 rx_bytes` and `net eth0 rx_packets` with scores ranging from 0.95 to 1 on 24 dates in February 2013, and (ii) correlations of `net eth0 rx_frame_errors` to `net ib0 rx_bytes` with a score of 1 on February 28 2013. We found that only the Spearman-Rank correlation method identified the correlated resource use counters on 25 dates. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one-hour.

Figure 5.6 shows the correlation of resource use counters that record cyclic redundancy check (CRC) errors and network data received by the Infiniband switch and compute nodes. We observe that there is a strong positive correlation between the resource use counters CRC errors and network data received. We identified: (i) correlations of `net ib0 rx_crc_errors` to `net eth0 rx_bytes` and `net eth0 rx_packets` with scores ranging from 0.95 to 1 on 24 dates in February 2013, and (ii) correlation between `net eth0 rx_crc_errors` and `net ib0 rx_bytes` with a score of 1 on February 28 2013. We found that only the Spearman-Rank correlation method identified the correlated resource use counters on 25 dates. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one-hour. Our results show that:

- The Infiniband switch received data frame errors and CRC errors on 24 out of

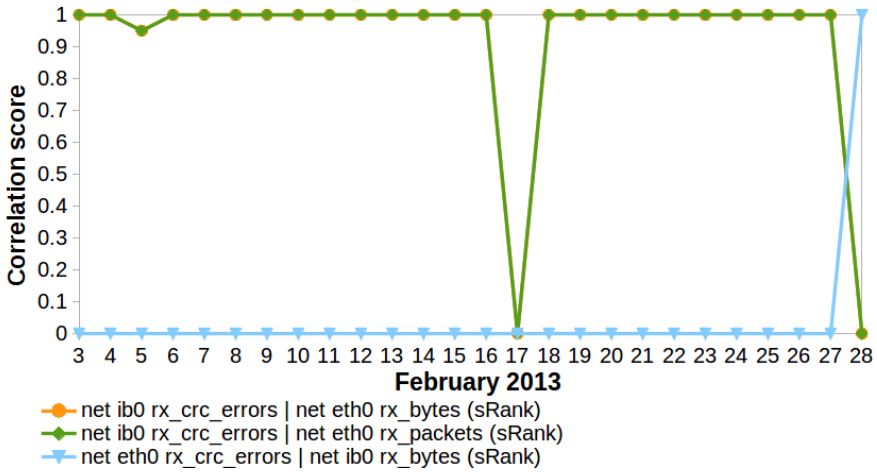


Figure 5.6: Correlation of cyclic redundancy check errors.

26 days in February 2013, representing 92% of the dates.

- The compute nodes received data frame errors and CRC errors on 1 out of 26 days in February 2013, representing 4% of the dates.
- The Spearman-Rank correlation method can be used as the primary method to identify network data errors. It identified patterns of network data errors that follow a monotonically increasing function.

Next, we will use the correlations of two different groups of error events to diagnose a networking software problem.

Phase 2: Correlated DNS Lookup Failure and GSIFTP Software Messages

The BIND (Berkeley Internet Name Domain) protocol is the most widely used DNS software on the Internet. The process by which one node locates another node on the basis of its name is specified in the DNS protocol. Usually the nodes on the network locate the destination node address and resolve its name successfully. In a rare occasion, the DNS software may be wrongly configured, for example a network supports only IPv4 addresses but the DNS software is configured for only IPv6 addresses. When the DNS software is configured with the wrong settings, then the node in the network is unable to locate the destination node address and resolve its name. When a DNS lookup failure occurs, an error message named `master network unreachable resolving` is recorded in the system logs. The error message `master network unreachable resolving` can be used to see what happens when a node attempts to locate another node on the network.

Figure 5.7 shows the correlation of DNS lookup failure and GSIFTP software messages.

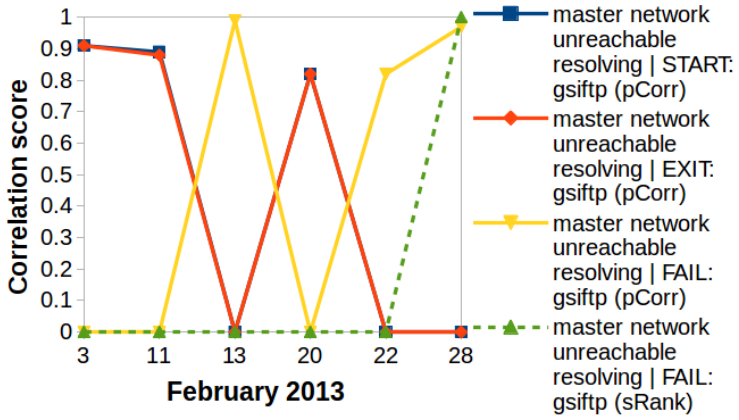


Figure 5.7: Correlation of DNS lookup failures & GSIFTP messages.

The messages `START: gsiftp`, `EXIT: gsiftp` and `FAIL: gsiftp` provide the state of the GSIFTP software. GSIFTP is the standard File Transfer Protocol enhanced with Grid Security Infrastructure (GSI). GSI is a specification for secret, tamper-proof, delegatable communication between software in a grid computing environment. We observe that there is a strong positive correlation between the DNS lookup failure and GSIFTP software state messages. We identified correlations of `master network unreachable resolving` to `START: gsiftp`, `EXIT: gsiftp` and `FAIL: gsiftp` with scores ranging from 0.82 to 0.99 on six days in February 2013. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one-hour. The dates of correlated DNS lookup failure and GSIFTP software messages coincide with the dates of correlated network data received and network data errors. We found that Pearson correlation identified the correlations on all six dates but Spearman-Rank correlation identified the correlations only on one date. If Pearson correlation alone identified the correlated DNS lookup failure and GSIFTP software messages on all the dates, then it can be used as the primary method. Our results show that Pearson correlation identified the correlated DNS lookup failures and GSIFTP software messages on all six days in February 2013. Hence, Pearson correlation can be used as the primary method.

Correlations with failures: From Figure 5.7, we observe that there is a strong positive correlation of DNS lookup failure message to the start (`START: gsiftp`) and exit (`EXIT: gsiftp`) status of the GSIFTP software on February 3, 11 and 20 2013. This showed that the DNS lookup failure did not cause the GSIFTP software to fail. Next, we scan the list of correlated events to determine the correlation strength between

`master network unreachable resolving` and `BUG: soft lockup`, `START: gsiftp` and `BUG: soft lockup`, and `EXIT: gsiftp` and `BUG: soft lockup`. We found that the DNS lookup failure, GSIFTP start and exit messages were weakly correlated to soft lockups on February 3 2013. Our result shows that DNS lookup failure and the GSIFTP software did not cause a compute node crash on Lonestar4. This represents a recovery rate of 100%.

From Figure 5.7, we observe that there is a strong positive correlation of DNS lookup failure to GSIFTP status fail (`FAIL: gsiftp`) messages on February 13, 22 and 28 2013. This showed that the GSIFTP software failed on three dates when DNS lookup failures occurred. We found that there are no soft lockup failures reported on February 13, 22 and 28 2013. Our result shows that a configuration error in BIND caused a DNS lookup failure which led to the GSIFTP software failure. The DNS lookup failure and GSIFTP software failure did not cause a compute node crash on Lonestar4.

Detailed diagnosis: When the GSIFTP software is executed, it uses the DNS software to locate the destination node and identify its name. The DNS software was unable to locate the destination node on six dates in February 2013. The DNS lookup failure generated an error message `master network unreachable resolving` which was recorded in the system logs. On three of the six dates, the DNS lookup failure occurred but the GSIFTP software started and exited normally. On the other three dates, the DNS lookup failure occurred and led to the GSIFTP software failure. Our result shows that the DNS software was unable to locate the destination node and resolve its name, which led to the GSIFTP software failure on those three dates. On the three dates, we found that the GSIFTP software failure did not cause a compute node crash on Lonestar4. This represents a recovery rate of 100%.

The benefit of combining analyses of Infiniband and compute node network interface counters and DNS lookup failures and FTP software errors is as follows: When on the same day, Infiniband and compute node network resource use counters are correlated and DNS lookup failure and FTP software messages are also correlated, it shows that network data transmission errors are associated with the generation of DNS lookup failure and network software messages. Therefore, we can use these correlations for monitoring the state of the network system.

Phase 3: Earliest Hour of Change

Table 5.5 shows the earliest hour of change in the correlated Infiniband and compute node network interface resource use counters and correlated DNS lookup failure and GSIFTP messages for the six dates on Lonestar4. On the six dates, we observe that the earliest hour of change is different. On four dates, the earliest hour of change

Table 5.5: Hours associated with the correlated data transmission error counters and correlated DNS lookup failures & GSIFTP error messages.

Lonestar4						
Correlated counters	**Feb 3**	**Feb 11**	**Feb 13**	**Feb 20**	**Feb 22**	**Feb 28**
Infiniband & compute node	10 AM	2 PM	2 AM	9 PM	3 AM	8 AM
Correlated errors	**Feb 3**	**Feb 11**	**Feb 13**	**Feb 20**	**Feb 22**	**Feb 28**
DNS lookup failures & GSIFTP errors	4 AM	9 PM	3 AM	3 PM	1 PM	12 PM

is associated with the correlated Infiniband and compute node network interface resource use counters. On two dates, the earliest hour of change is associated with the correlated DNS lookup failure and GSIFTP messages. If the correlated resource use counters were used as the only data source, then the earliest hour of change on two dates would not be identified. If the correlated errors were used as the only data source, then the earliest hour of change on four dates would not be identified. Our results show that we require both the correlated resource use counters and correlated errors to identify the earliest hour of change on all six dates. We found that there are different time windows between the hour of change on all six dates. The time window ranges from one hour to 10 hours.

Validation

Next, we test the significance of the correlation coefficients of: (i) groups of resource use counters that are strong positive correlated, and (ii) groups of errors that are strong positive correlated. We tested all the correlation coefficients against the null hypothesis. We obtain the z-scores for all the correlation coefficients and a summary is provided in Table 5.6.

Table 5.6: Summary of z-scores. n_r contains the number of hours in one day of resource usage logs. n_e contains the number of hours in one day of system logs.

Lonestar4						
Correlated counters	**Feb 3**	**Feb 11**	**Feb 13**	**Feb 20**	**Feb 22**	**Feb 28**
Infiniband & compute node ($n_r = 24$)	$z_r =$ 12.13	$z_r =$ 12.13	$z_r =$ 12.13	$z_r =$ 12.13	$z_r =$ 12.13	$z_r =$ 12.13
Correlated errors	**Feb 3**	**Feb 11**	**Feb 13**	**Feb 20**	**Feb 22**	**Feb 28**
DNS lookup failures & GSIFTP errors	$z_e =$ 5.23	$z_e =$ 5.06	$z_e =$ 12.13	$z_e =$ 4.02	$z_e =$ 4.02	$z_e =$ 8.35

From Table 5.6, we observe that the z-scores for all the correlation coefficients range from 4.02 to 12.13. At the 99% confidence level, under the null hypothesis $z_{0r} = 2.64$ and $z_{0e} = 2.64$. Hence, we reject the null hypothesis in favour of the alternate hypothesis.

Next, for all hypothesis tests we use the significance level, $\alpha = 0.01$ and apply a one-sided test to obtain a P-value. We use the P-value to determine the probability of rejecting the null hypothesis when it is true. Table 5.6 summarises the z-scores for all the correlation coefficients. We observe that the smallest z-score is 4.02. Since this is a one-sided test, the P-value is equal to the probability of observing a value greater than 4.02 in the standard normal distribution, or $P(Z > 4.02) = 1 - P(Z \leq 4.02) = 1 - 0.99995 = 0.00005$. To account for inflation in false positive due to multiple independent tests, we obtain the adjusted P-value is $0.00005 \times 26 = 0.0013$ where 26 is the total number of days. The P-value is less than 0.01, indicating it is highly unlikely this result would be observed under the null hypothesis. All the z-scores are greater than or equal to 4.02. Therefore, the adjusted P-value for all the z-scores are less than 0.01, indicating it is highly unlikely these results would be observed under the null hypothesis.

5.3.2 Storage System and Linux Processes

In this error case, we determine: (i) the correlations of harddisk, filesystem and Linux processes resource use counters, and (ii) the correlations of filesystem, Linux process and software errors. We use the correlations to diagnose problems in the storage system. Then, we assess the system reliability.

Phase 1: Correlated Harddisk, Filesystem and Linux Process Resource Use Counters

When a Linux process sends data to be read or written to the harddisk, three steps are executed to perform the operation. In the first step, the filesystem allocates an inode. TACC_Stats increments the resource use counter named `llite /work alloc_inode` which records the number of inodes allocated on the work partition of the filesystem. In the second step, it performs a seek operation. TACC_Stats increments the resource use counter named `llite /work seek` which records the number of seek operations on the work partition on the filesystem. In the third step, the data is written to the harddisk. TACC_Stats increments the resource use counter named `md0 wr_sectors` which records the number of sectors data is written to the harddisk. The resource use counter named `ps processes` records the number of Linux process created. The resource use counter named `ps ctxt` records the number of context switches across

all the CPUs. The harddisk, filesystem and Linux process resource use counters can be used to see what happens when Linux processes write to the storage system.

Figure 5.8 shows the correlation of Linux process and filesystem resource use counters.

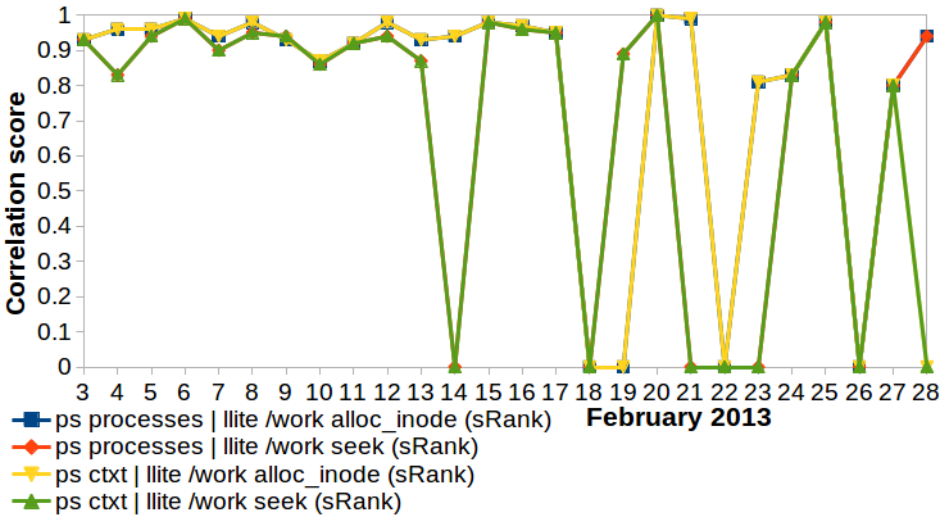


Figure 5.8: Correlation of Linux process & filesystem resource use counters.

We observe: (i) strong positive correlations of `ps processes` and `ps ctxt` to `llite /work alloc_inode` with scores ranging from 0.8 to 1 on 21 dates, and (ii) strong positive correlations of `ps processes` and `ps ctxt` to `llite /work seek` with scores ranging from 0.8 to 1 on 19 dates. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one hour. We found that only the Spearman-Rank correlation method identified the correlations on all the dates.

Figure 5.9 shows the correlation of Linux process, harddisk and filesystem resource use counters. We observe: (i) strong positive correlations of `md0 wr_sectors` to `llite /work alloc_inode` and `llite /work seek` with scores ranging from 0.8 to 1 on 18 dates, and (ii) strong positive correlations of `md0 wr_sectors` to `ps processes` and `ps ctxt` with scores ranging from 0.95 to 1 on 24 dates. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we focus on the correlations obtained on time-bins of one hour. We found that only the Spearman-Rank correlation method identified the correlations on all the dates. Our results show that:

- When a Linux process writes to the storage system, a strong correlation of

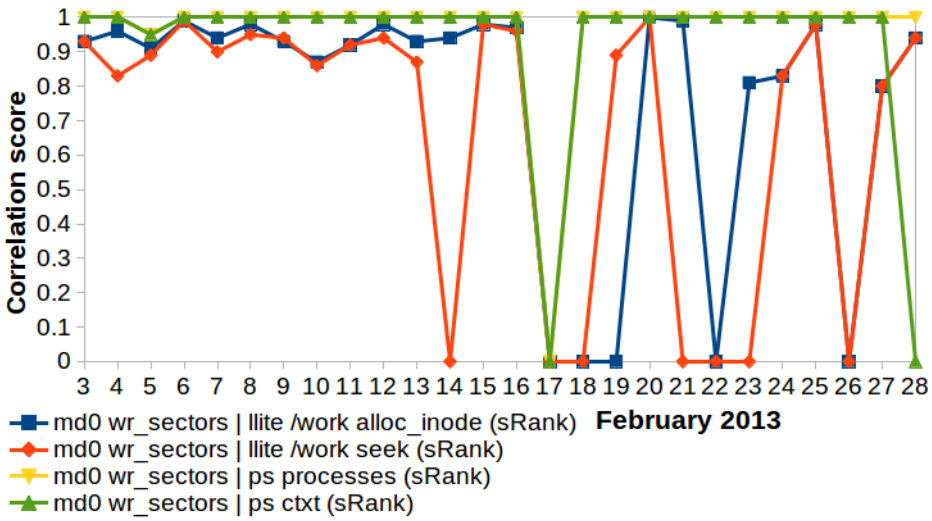


Figure 5.9: Correlation of harddisk, filesystem & Linux process resource use counters.

harddisk and Lustre filesystem activities is generated.

- The Spearman-Rank correlation method can be used to identify activities on the harddisk and Lustre filesystem. It identified patterns of harddisk, filesystem and Linux process that follow a monotonically increasing function.

Phase 2: Correlated Filesystem, Process and Software Errors

We identify a filesystem I/O error by searching the system logs for a message that contains the keywords `Filesystem: xfs_log _force: error returned`. A harddisk can degrade when it is heavily used over a period of time. When a harddisk starts to fail, it will produce bad sectors. If a bad sector is detected on a harddisk, the bad sector will be marked and the filesystem is informed not to write to that sector. In most cases, the filesystem will locate a good sector on the harddisk and execute its I/O operation successfully. However, in a rare occasion, if the filesystem is unable to locate a good sector on the harddisk, then it fails to execute its I/O operation.

From Figure 5.10, we observe: (i) a strong positive correlation of `Filesystem: ...: error returned` to a process message `Pid: comm calc_du` with a score of 0.99 and `BUG: soft lockup` message with a score of 0.98 on February 3 2013, and (ii) a strong positive correlation between `Pid: comm calc_du` and `BUG: soft lockup` with a score of 0.99 on February 3 2013. The dates of the correlated errors coincide with the dates of correlated Linux processes, harddisk and filesystem I/O resource use counters. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. Therefore, we

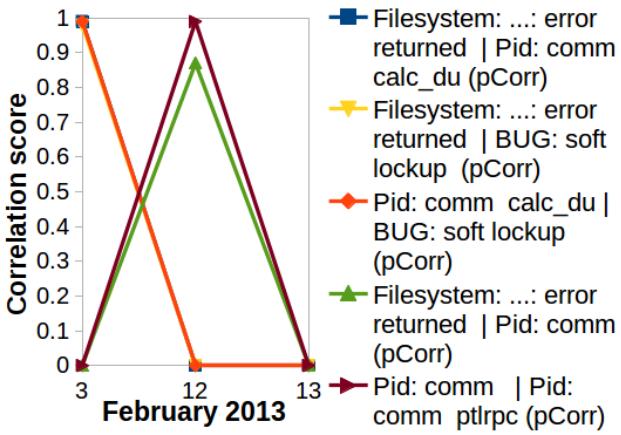


Figure 5.10: Correlation of filesystem error & process information messages.

focus on the correlations obtained on time-bins of one hour. We found that only the Pearson correlation method identified the correlated filesystem error, process and soft lockup messages. Our results show that:

- Filesystem I/O errors are associated with process errors and soft lockup events as observed from the correlations of `Filesystem: ...: error returned`, `Pid: comm calc_du` and `BUG: soft lockup` messages shown in Figure 5.10.
- The Pearson correlation method can be used to identify the correlated filesystem error, process and soft lockup messages. It identified patterns of filesystem error, process information and soft lockup messages that follow a linear function.

From Figure 5.10, we observe: (i) a strong positive correlation between `Filesystem: ...: error returned` and `Pid: comm` with a score of 0.87 on February 12 2013, and (ii) a strong positive correlation between `Pid: comm` and `Pid: comm ptlrpc` with a score of 0.99 on February 12 2013. The date of the correlated errors coincide with the date of correlated harddisk, filesystem I/O and Linux process resource use counters. There are small changes in the correlation scores obtained on time-bins of 20 and 40 minutes but the correlation scores are within 0.8 to 1. We found that only the Pearson correlation method identified the correlated filesystem error and process message. Hence, Pearson correlation can be used as the primary method. There are no soft lockup messages reported on February 12 2013.

From Figure 5.11, we observe that there is a strong positive correlation of the error message `Filesystem: ... : error returned` to: (i) two software error messages, (ii) one filesystem recovery protocol message, and (iii) one memory leak error message. The correlation scores range from 0.81 and 0.97 on four dates. The dates of the correlated errors coincide with the dates of correlated harddisk, filesystem

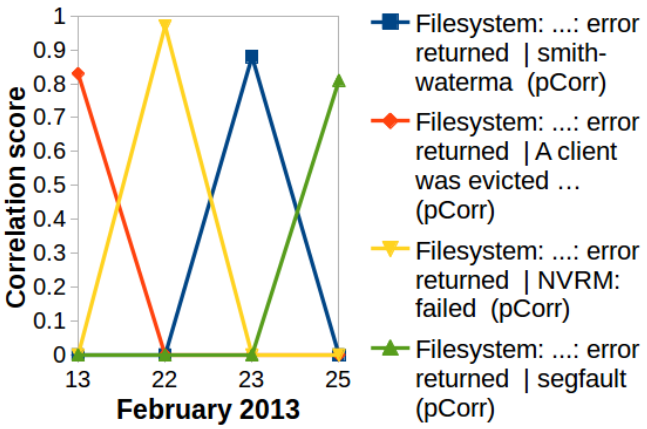


Figure 5.11: Correlation of filesystem & software error messages.

I/O and Linux processes resource use counters. The software error messages contain the keywords `smith-waterman` and `NVRM: failed`. The filesystem recovery protocol message contains the keywords `a client was evicted` and the memory leak error message contains the keyword `segfault`. `smith-waterman` is associated with the smith-waterman algorithm that performs local sequence alignment for determining similar regions between two strings of nucleic acid sequences or protein sequences. `NVRM: failed` is associated with the NVIDIA graphics card. The message indicated that the NVIDIA graphics card driver failed. `a client was evicted` is associated with the Lustre filesystem Evict/RPC protocol. The message indicated that the Lustre filesystem removed an unresponsive client. The error message `segfault` indicated that a program suffered a memory leak.

Correlation with failures: There were no soft lockup events reported on February 13, 22 and 25 2013. Soft lockup events were reported on February 23 2013. Next, we manually scan the list of correlated events obtained for February 23 2013 to determine if the filesystem error and software error messages are strongly correlated to soft lockup events. We found that `Filesystem: ...: error returned` and `smith-waterman` were weakly correlated to soft lockup events on February 23 2013. We found that only the Pearson correlation method identified the correlated errors on all the dates. Hence, Pearson correlation can be used as the primary method.

Detailed diagnosis: When the filesystem encountered an I/O error, the process or software that was writing files to the filesystem generated an error message. We showed that the filesystem error messages are strongly positive correlated to process and software error messages. Our result shows that when a filesystem I/O error occurred, the process and software were unable to obtain I/O access which led to process and software errors. On five out of six days in February 2013, the filesystem recovered from the I/O errors. This represents a recovery rate of 83%. However, on

February 3 2013 the filesystem failed to recover from the I/O error which led to a compute node soft lockup. This represents a failure rate of 17%.

The benefit of combining analyses of harddisk, filesystem I/O and Linux processes resource use counters with filesystem I/O, process and software errors is given as follows: When correlations of harddisk, filesystem I/O and Linux processes resource use counters and correlations of filesystem I/O, process and software errors occur on the same day, it shows that harddisk, filesystem I/O and Linux processes activities are associated with the generation of filesystem I/O, process and software errors. Therefore, we can use the correlations to monitor the health of the storage system.

Phase 3: Earliest Hour of Change

Table 5.7 shows the earliest hour of change in the correlated harddisk, filesystem I/O and Linux processes resource use counters and correlated filesystem I/O, process and software errors.

Table 5.7: Hours associated with the correlated harddisk, filesystem I/O and Linux processes resource use counters and correlated filesystem I/O, process and software errors.

Lonestar4						
Correlated counters	**Feb 3**	**Feb 12**	**Feb 13**	**Feb 22**	**Feb 23**	**Feb 25**
HDD, filesystem & processes	5 AM	1 AM	4 AM	4 PM	3 PM	7 AM
Correlated errors	**Feb 3**	**Feb 12**	**Feb 13**	**Feb 22**	**Feb 23**	**Feb 25**
Filesystem, process & software errors	4 AM	12 AM	3 AM	1 PM	12 PM	12 PM

On all the dates, we observe that the earliest hour of change in the system behaviour is different. On February 25 2013, the earliest hour of change is associated with the correlated harddisk, filesystem I/O and Linux processes resource use counters. On five dates, the earliest hour of change is associated with the correlated filesystem I/O, process and software errors. If we use only the correlated harddisk, filesystem I/O and Linux processes resource use counters, then the earliest hour of change in the system behaviour on five dates would not be identified. Having said that, if we use only the correlated filesystem I/O, process and software errors, then the earliest hour of change in the system behaviour on February 25 2013 would not be identified. Our results show that we require both the correlated resource use counters and correlated errors to identify the earliest hour of change in the system behaviour on

all six dates. We found that there are different time windows between the hour of change on all six dates. The time window ranges from one to five hours.

Validation

Next, we test the significance of the correlation coefficient of: (i) groups of resource use counters that are strong positive correlated, and (ii) groups of error events that are strong positive correlated. We tested all the correlation coefficients against the null hypothesis. We obtained the z-scores for all the correlation coefficients and a summary is given in Table 5.8.

Table 5.8: Summary of z-scores. n_r contains the number of hours in one day of resource usage logs. n_e contains the number of hours in one day of system logs.

Lonestar4						
Correlated counters	**Feb 3**	**Feb 12**	**Feb 13**	**Feb 22**	**Feb 23**	**Feb 25**
HDD, filesystem & processes ($n_r = 24$)	$z_r =$ 6.25	$z_r =$ 6.56	$z_r =$ 4.86	$z_r =$ 12.13	$z_r =$ 3.84	$z_r =$ 9.49
Correlated errors	**Feb 3**	**Feb 12**	**Feb 13**	**Feb 22**	**Feb 23**	**Feb 25**
Filesystem, process & software errors ($20 \leq n_e \leq 24$)	$z_e =$ 9.55	$z_e =$ 4.72	$z_e =$ 4.08	$z_e =$ 8.61	$z_e =$ 4.98	$z_e =$ 3.75

From Table 5.8, we observe that the z-scores for all the correlation coefficients range from 3.75 to 12.13. At the 99% confidence level, under the null hypothesis $z_{0r} = 2.64$ and $z_{0e} = 2.64$. Hence, we reject the null hypothesis in favour of the alternate hypothesis.

Next, for all hypothesis tests we use the significance level, $\alpha = 0.01$ and apply a one-sided test to obtain a P-value. We use the P-value to determine the probability of rejecting the null hypothesis when it is true. We observe that the smallest z-score is 3.75. Since this is a one-sided test, the P-value is equal to the probability of observing a value greater than 3.75 in the standard normal distribution, or $P(Z > 3.75) = 1 - P(Z \leq 3.75) = 1 - 0.9999 = 0.0001$. To account for inflation in false positive due to multiple independent tests, we obtain the adjusted P-value $0.0001 \times 26 = 0.0026$, where 26 is the number of dates. The P-value is less than 0.01, indicating it is highly unlikely this result would be observed under the null hypothesis. We observe that all the z-scores are greater than or equal to 3.75. Therefore, the adjusted P-value for all the z-scores are less than 0.01, indicating it is highly unlikely these results would be observed under the null hypothesis.

5.4 Summary

In this chapter, we applied CORRMEXT on multiple HPC systems. CORRMEXT generated the analyses focused on the correlated groups of resource use counters and correlated groups of errors, and diagnosed three new error cases and one resource activity case. It identified the earliest occurrences of the problem by extracting the variance in the times of the correlated resource use counter groups and correlated error groups. CORRMEXT applied Fisher's z-score and the Bonferroni correction and showed that all the correlations are significant.

We provide our recommendation on what one should look for in the resource use data and system logs. In the resource use data, we can use the following types of resource use counters to do the following: (i) bytes received on the network, number of network packets transmitted, number of network data frame errors and number of network data CRC errors to identify network data errors, (ii) number of Linux processes created, number of context switching, number of inodes allocated, number of filesystem seek operations and number of harddisk sector writes to monitor activities between Linux processes, harddisk and filesystem.

In the system logs, we can use the following types of messages to do the following: (i) DNS lookup failure, FTP software start, FTP software exit and FTP software failure to identify a configuration error in the DNS software, (ii) filesystem I/O error, process information and software error to identify a cause of filesystem I/O execution failure.

Chapter 6

A Features Correlation-based Workflow for HPC Systems Diagnosis

There is little work which show that multiple feature extraction methods are required to identify more system messages and resource use counters associated with rare error cases. In this chapter, we present a new workflow that combines resource use data matrices and message types data matrices for identifying rare error cases. We named the workflow EXERMEST (**EX**tracting F**E**atures and Co**R**relating Resource Use Counters and **MES**sage **T**ypes). EXERMEST extracts the nodes associated with the identified errors. There are no message types data matrices available on Stampede-1. Therefore, we focus on the resource use data matrices and message types data matrices available on Ranger and Lonestar4.

We structure this chapter as follows: In Section 6.1, we introduce the EXERMEST framework. In 6.2, we describe the system models, problem specification and details for the modules used within the EXERMEST framework. In 6.3, we evaluate the feature extractors and present our analyses from EXERMEST for a series of error cases on the Ranger and Lonestar4 HPC systems and conclude with a summary in 6.4.

6.1 Introduction

There is a large body of research on detecting errors [16, 33, 34] and diagnosing failures [8–10, 72] that showed that combining system logs and resource use data can improve error detection and failure diagnosis over using system logs alone. The methods developed by N. Gurumdimma et. al. [33, 34] focus on improving

error detection. Their method presented in [33] showed that the error handling time window can be increased by up–to 50 minutes by combining system logs with resource use data. An approach called CRUDE [34] showed that the error detection accuracy can be improved, on average by 85% over NodeInfo [62], by combining resource use data and system logs. The technique developed by Z. Zheng et. al. [71] focus on identifying characteristics of system failures. They combine RAS and Jobs logs on a HPC system to identify interesting failure characteristics. The ANCOR framework [8] developed a two-phase approach where: (i) in the first phase, anomalous nodes are identified on resource use data to provide a partial diagnosis, and (ii) in the second phase, errors that are correlated to system failures on the anomalous nodes are identified on system logs to provide a more detailed diagnosis. A tool called LogAider that is developed by S. Di et. al. [16] focus on identifying error propagation that lead to system failures. They combine RAS and Job logs to identify correlations of system events and failures across space and time on a HPC system. The diagnostics frameworks reported in references [8, 9, 16, 72] have correlated errors to system failures only.

The anomaly detection frameworks reported in references [8, 27, 34] have applied only Principal Component Analysis to identify the significant system metrics in resource usage data. The anomaly detection framework reported in reference [44] have evaluated PCA-based and ICA-based anomaly detection. They showed that ICA-based anomaly detection is more effective than PCA-based anomaly detection. However, there is little work which show that multiple feature extraction methods are required for identifying more system messages and resource use counters associated with rare error cases. To bridge this gap, we implement and evaluate a new workflow – called EXERMEST – that combines resource use data matrices with message types data matrices. EXERMEST identifies significant resource use counters and messages. In the EXERMEST workflow, multiple feature extraction algorithms are evaluated. We implemented a two-phase approach where: (i) in the first phase, different feature extractors are applied to identify significant errors and resource use counters, (ii) in the second phase, the significant resource use counters are correlated to other resource use counters and the significant system events are correlated to other events. Then, the significance of all the correlation coefficients are validated. By significant, we mean resource use counters and system events which are assigned highest scores by the feature extractors.

6.1.1 Contributions

The main contributions of this chapter include:

- A new workflow called EXERMEST that combines resource use data matrices and message types data matrices for HPC systems diagnosis. EXERMEST identifies significant system messages and resource use counters associated with rare error cases.

- A demonstration that EXERMEST improves failure diagnosis over previous research. EXERMEST show that multiple feature extraction methods are required to identify the significant system messages and resource use counters associated with the rare error cases.

- Identification of: (i) significant Infiniband network packet drops and Lustre filesystem I/O resource use counters that are correlated to Lustre filesystem I/O errors, (ii) significant CPU I/O and Lustre filesystem I/O resource use counters that are correlated to hard disk and Lustre filesystem I/O errors, (iii) significant Linux virtual memory resource use counters and Lustre filesystem I/O resource use counters that are correlated to Linux memory management errors, and (iv) significant Linux process threads and dirty memory resource use counters that are correlated to data and network synchronisation errors. The Infiniband network packet drop, Lustre filesystem I/O, CPU I/O, Linux virtual memory, Linux threads and dirty memory resource use counters are potential flags for online detection of Lustre filesystem I/O errors, hard disk I/O errors, Linux memory management errors and data and network synchronisation errors.

- That 10 minutes, 30 minutes and 1-hour time-bins are required for identifying the correlated system errors and correlated resource use counters.

- A detailed statistical validation step to ensure an accurate assessment of the diagnosis. EXERMEST uses the Bonferroni correction to show that it is highly unlikely all the correlations would be observed under the null hypothesis.

6.2 System Models, Problem Specification and EXERMEST Framework

The system model to which the EXERMEST framework can be applied is described in Chapter 3.1; we summarise the system model here. A HPC system is comprised of a job scheduler, system software stack and sets of nodes and jobs. The job scheduler, nodes, jobs and system software stack generate system logs - they contain system events and failure data. The nodes, jobs and system software stack generate resource usage logs - they currently contain 410 system resource use counters (refer to Table

3.1). The system and resource use logs are archived on a centralised message logging system.

6.2.1 Significant Errors on Nodes

Let S be a HPC system with n distinct nodes and all the nodes in the system S are linked together by a network. We capture the significant errors by first defining a set $NS_i = \{n_1, n_2, ..., n_i | n_i \in Node_{state}\}$, where $1 \leq i < x$, x is the number of nodes, $Node_{state} = \{n_{fault}, n_{error}, n_{failed}\}$, n_{fault} = a node on which a fault is triggered, n_{error} = a node that contained an error, n_{failed} = a node that crashed. Thus, an error can occur within one node or occur on two or more nodes.

Due to the possibility that errors are associated with a large number of nodes, finding the significant errors is challenging. In this chapter, we will present the EXERMEST framework that investigates the use of feature extraction methods to identify the significant errors and the nodes associated with the identified errors.

6.2.2 Problem Specification

The problem we address in this chapter is specified in [12] and we describe the problem here: Given (i) a set of resource use data matrices, (ii) a set of message types data matrices, (iii) a list of resource use counters, (iv) a list of message types, and (v) the number of dates, then:

1. Identify the significant resource use counters by time-bins,
2. Identify the significant message types by time-bins,
3. Identify resource use counters that are strongly linearly or monotonically correlated to the significant counters by time-bins on the specified dates,
4. Identify messages that are strongly linearly or monotonically correlated to the significant messages by time-bins on the specified dates.

A time-bin is one window of a fixed time interval. The number of resource use and message types data matrices that we collect are based on the given number of dates. We develop the *EXERMEST* (**EX**tracting F**E**atures and Co**R**relating Resource Use Counters and **MES**sage **T**ypes) framework as shown in Figure 6.1. EXERMEST applies the Features extraction and Correlation modules on the resource use and message types data matrices. Each module produces a set of diagnostics reports. The reports can be used to identify significant error propagation and recovery cases. EXERMEST is available for downloading at `https://diag-toolkits.github.io/EXERMEST/`. Next, we describe the two modules used within the EXERMEST framework.

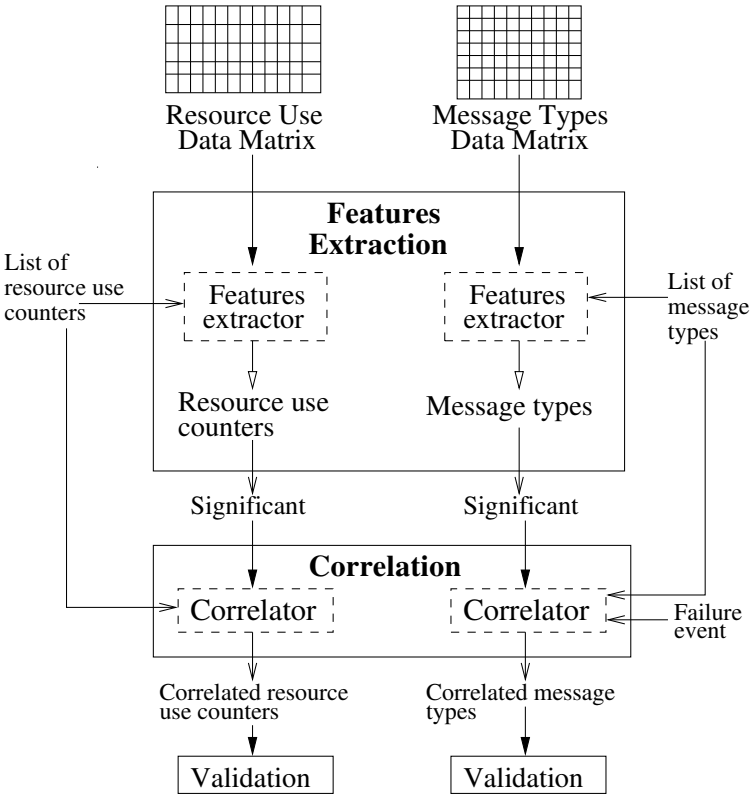


Figure 6.1: The workflow of the EXERMEST framework.

6.2.3 EXERMEST: Feature Extraction

The Feature Extraction module is presented in [12] and we describe the details here. A resource use log is comprised of a number of attributes and an example is given as follows:

```
204865 Jul 12 12:30:01 i132-206 eth0 rx_bytes 352894 ..
```

In the resource use log, each attribute is separated by a white space. The first attribute represents a job (`204865`). The second, third and fourth attributes represent the month, date and time (`Jul 12 12:30:01`) the resource use log was generated. The fifth attribute represents a node (`i132-206`) on the HPC system. The sixth attribute represents the device name (`eth0`); in this example, the device is a network interface card. The remaining pairs of attributes represent a resource use counter (e.g., `rx_bytes`) and its value (`352894`).

To generate the resource use data matrix, we implemented a process that extracts the resource use counters and their values in the resource use logs. The process is described in Chapter 3.4.1 and we summarise it here. The process works as follows:

- We divide the resource use log into time-bins of 1 hour, 30 and 10 minutes by the given date.
- We extract all the resource use counters in the resource use log and store the resource use counters in a list.
- We identify the unique resource use counters and store them in a list of (unique) resource use counters.
- We match a resource use counter in the list of unique resource use counters to the resource use counter in the resource use log and obtain the value of the resource use counter between two consecutive resource use logs. We obtained the values of the resource use counters separately by 1 hour, 30 and 10 minute time-bins.

A system log is comprised of a number of attributes and an example is given as follows:

```
227893 Jul 18 08:43:10 i172-108 kernel LustreError: connection
restored
```

In the system log, each attribute is separated by a white space. The first attribute represents the job (`227893`). The second, third and fourth attributes represent the month, date and time (`Jul 18 08:43:10`) the system log was generated. The fifth attribute represents a node (`i172-108`) on the HPC system. The sixth attribute represents the system component (`kernel`); in this example, the system component is the Linux operating system kernel. The remaining white space separated attributes represent the system event. In this system event, the Lustre filesystem had restored its connection (`LustreError: connection restored`).

To generate the message types data matrix, we implemented a process that extracts the system events and their counts in the system logs. The process is described in Chapter 3.4.2 and we summarise it here. The process works as follows:

- We divide the system logs into time-bins of 1 hour, 30 and 10 minutes by the given date.
- We extract all the system events in the system logs and store the system events in a list.
- We identify the unique system events and store them in a list of message types.
- We count the message types separately by 1 hour, 30 and 10 minute time-bins.

Currently, we evaluate three feature extraction algorithms. The algorithms are: (i) Principal Component Analysis (PCA), (ii) Independent Component Analysis (ICA), and (iii) Non-linear Principal Component Analysis (NLPCA). We use PCA and ICA for identifying resource use counters and message types which are *linearly* uncorrelated. We use NLPCA for identifying resource use counters and message types which are *non-linearly* uncorrelated. PCA, ICA and NLPCA are unsupervised dimensionality-reduction methods that do not require a-priori knowledge about the data labels. Therefore, they can be used for identifying the significant resource use counters and message types without the need to label the data. We integrate the PCA, ICA and NLPCA methods into the Features Extraction module. Specifically, the algorithms we use are: (i) the robust PCA algorithm [13] for obtaining the PCA components, (ii) the fast ICA algorithm [39] for obtaining the ICA components, and (iii) the NLPCA algorithm that is based on a neural network [60] for obtaining the NLPCA components. A large HPC system can monitor hundreds of different resource use counters and it can generate thousands of different message types. To identify the significant resource use counters and message types is an essential but non-trivial task [28, 44, 52].

The Feature Extraction module receives as its input, a resource use data matrix and a message types data matrix as shown in Figure 6.1. The resource use data matrix RUD_t has m rows and n columns. Each row $m_i \in RUD_t$ represent one resource use counter, each column $n_j \in RUD_t$ represent one time-bin and each cell $mc_{ij} \in RUD_t$ contains the count of resource use counter m_i at time-bin n_j. The message types data matrix MTD_t has x rows and y columns. Each row $x_i \in MTD_t$ represent one message type, each column $y_j \in MTD_t$ represent one time-bin and each cell $mc_{ij} \in MTD_t$ contains the count of message type x_i at time-bin y_j.

PCA Features Extractor

PCA converts the set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. Because the rows of the resource use and message types data matrices are the features and their columns are the time-bins, first we transpose the data matrices RUD_t and MTD_t to obtain RUD_t' and MTD_t'. The resource use counters collected by TACC_Stats range from CPU usage to memory, Lustre I/O, network, virtual memory, process and NUMA counters. The scale of the data collected may be different. For example, CPU utilisation values can be given in percentage and Lustre I/O, memory, network, virtual memory, process and NUMA counter values can be actual counts. To solve the problem of different scales in the data, we normalise the values for all the resource use counters such that their values range between 0 and 10. To capture the true

variance, we adjust the columns in the normalised resource use data matrices to have zero mean. To identify the significant message types under identical conditions, we normalise the message types values and adjust the columns in the normalised data matrices so that its columns have zero mean.

Next, we calculate the covariance matrices C_{RUD} and C_{MTD} using the normalized zero-mean matrices $RUD_t'^{N0}$ and $MTD_t'^{N0}$. The covariance matrix has entries c_{ij} defined as [1]: $cov(X,Y) = \frac{1}{n^2}\sum_i \sum_{j>1}(x_i - x_j).(y_i - y_j)$ where X and Y are variables that can take on the values (x_i, y_i) for $i = 1, ..., n$, $x_i \in X$ and $y_i \in Y$. We obtain: (i) the resource use data covariance matrix $C_{RUD} = \frac{1}{n-1} RUD_t'^{N0} RUD_t''^{N0}$ where $RUD_t''^{N0}$ is the transpose of $RUD_t'^{N0}$, and (ii) the message types covariance matrix $C_{MTD} = \frac{1}{n-1} MTD_t'^{N0} MTD_t''^{N0}$ where $MTD_t''^{N0}$ is the transpose of $MTD_t'^{N0}$. Currently, our PCA features extractor uses the robust PCA algorithm [13] to obtain the principal components. The reference in [44] have shown that the first principal component contains the largest variance. Therefore, we extract the scores of the supplied data on the first principal component.

ICA Features Extractor

Given the normalized zero-mean data matrices $RUD_t'^{N0}$ and $MTD_t'^{N0}$, ICA also finds a new set of values of linearly uncorrelated variables. Differently to PCA, the components identified by ICA are not necessarily orthogonal. A pre-processing step called whitening [42] is first applied on the input data matrix to obtain a new set of variables which are uncorrelated and each have a variance of 1. Whitening the input data matrix ensures that the average covariance between the whitened variables and original variables is maximal [42]. Then, the ICA algorithm is applied on the whitened data matrix. Currently, our ICA features extractor uses the fast ICA algorithm [39] to obtain the ICA components. Based on the reference in [44] that show that the first principal component contains the largest variance, we extract the estimated scores of the supplied data on the first ICA component.

NLPCA Features Extractor

Given the normalized zero-mean data matrices $RUD_t'^{N0}$ and $MTD_t'^{N0}$, NLPCA finds a new set of values of uncorrelated variables. Differently to PCA, the principal components identified by NLPCA are non-linear (i.e., curved). Currently, our NLPCA feature extractor uses a neural network based algorithm [60] that provides a non-linear model of the mapping function to obtain the principal components. Based on the reference in [44], we extract the scores of the supplied data on the first component.

Extracting the Significant Features

We implement a process for extracting the score and its associated resource use counter and message type. The process works as follows: Let L_{pair} be a list that contains pairs of $\langle index, score \rangle$ where $index$ is the index of a resource use counter or message type, $score$ is the score of the resource use counter or message type and $Size_L = |L_{pair}|$ is the total number of index-score pairs in the list. First, we sort the list L_{pair} in descending order such that the first pair $\langle index_1, score_1 \rangle$ is the resource use counter or message type with the highest score. Then, we extract a subset $L_{pair}^{subset} \subset L_{pair}$ such that $\forall \langle index, score \rangle \in L_{pair}^{subset}$, $score > 0$ and $|L_{pair}^{subset}| = \lceil 0.05 \times Size_L \rceil$. Next, $\forall \langle index, score \rangle \in L_{pair}^{subset}$ we map $index$ to the list of resource use counters and list of message types and obtain the list of significant resource use counters and list of significant message types. Algorithms such as PICK [4] can also be used to select the top 5% of resource use counters and message types.

6.2.4 EXERMEST: Correlation

The Features Extraction module described in the preceding section extracted the lists of significant resource use counters and message types. Then, these lists are given to the Correlation module to correlate the significant resource use counters to other counters and correlate the significant message types to other message types. The Correlation module is presented in [12] and we describe it here. Specifically, the Correlation module performs the following:

- It calculates the correlation coefficient for all pairs of significant resource use counters and other resource use counters, and obtain a resource use counters correlation matrix.
- It calculates the correlation coefficient for all pairs of significant message types and other message types, and obtain a message types correlation matrix.

To obtain the resource use counters and message types correlation matrices, we implemented a process that works as follows:

- We obtain the significant data matrices containing counts of all the significant resource counters or message types by time-bins of 1 hour, 30 and 10 minutes.
- We obtain the data matrices containing counts of all the resource counters or message types by time-bins of 1 hour, 30 and 10 minutes.
- We apply two correlation methods on the significant resource use data matrix and resource use data matrix by time-bins of 1 hour, 30 and 10 minutes, and

obtain the resource use counters correlation matrices by 1 hour, 30 and 10 minute time-bins.

- We apply two correlation methods on the significant message types data matrix and message types data matrix by time-bins of 1 hour, 30 and 10 minutes, and obtain the message types correlation matrices by 1 hour, 30 and 10 minute time-bins.

In our Correlation module, we currently use Pearson correlation and Spearman-Rank correlation methods [1]. The details are described in Chapter 4.3.2 and we provide a summary here. Pearson correlation coefficient [1], r is defined as the mean of the products of the standard scores, $r = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{x_i-\bar{x}}{s_x}\right)\left(\frac{y_i-\bar{y}}{s_y}\right)$ where $\left(\frac{x_i-\bar{x}}{s_x}\right)$ is the standard score of x, $\left(\frac{y_i-\bar{y}}{s_y}\right)$ is the standard score of y, x and y are two datasets containing n values of a pair of resource use counters or a pair of message types, $s_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2}$ is the sample standard deviation of x, $s_y = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i-\bar{y})^2}$ is the sample standard deviation of y, $\bar{x}$ and $\bar{y}$ is the sample mean of x and y. We implemented the Pearson correlation algorithm to identify resource use counters and message types which are correlated linearly.

Spearman-Rank correlation coefficient [1], ρ is defined as the Pearson correlation coefficient between the ranks of a pair of variables. The following is assumed by Spearman-Rank correlation: (i) the value of a variable increases when the value of another variable increases, (ii) the value of a variable remains when the value of another variable remains. We implemented the tied rank average method [67] and obtain the ranked values for all pairs of variables. After we obtained the ranked values for all pairs of variables, we input the ranked values into Pearson correlation method to obtain Spearman-Rank correlation coefficient. We implemented the Spearman-Rank correlation algorithm to identify resource use counters and message types which are correlated monotonically.

There are many methods available. However, those methods assume that the data variables are i.i.d, i.e., each variable has the same probability distribution as all the other variables, and all the variables are mutually independent. The references in [24, 51] have shown that correlation algorithms are effective in identifying influence between interacting system components [51] and modeling system behaviour [24]. While a HPC system generates resource utilisation data at regular intervals, it generates error and failure messages only when the error output statement in the program code is executed. Because of this, the timestamps in the resource use data and system logs are different. The correlation algorithms require that the same number of data points is available on the x-axis in the dataset in order to calculate the correlation coefficient. As such, we do not identify correlations of resource use

counters to message types. Having said that, our objective is to identify correlations of resource use counters and correlations of message types. We use the following rules for interpreting the strength of the correlation coefficients [1]: (a) 0.8 to 1: Strong positive correlation, (b) 0.3 to 0.79: Moderate positive correlation, (c) 0.1 to 0.29: Weak positive correlation.

Testing the Significance of the Correlation Coefficients

The technique we used to test the significance of all the correlation coefficients is presented in Chapter 4.3.2; we summarise the validation technique here. We define two null hypotheses and two alternate hypotheses. The null hypotheses are: (i) H_{0r} that two resource use counters are positive correlated with a score between 0.1 and 0.29, and (ii) H_{0m} that two message types are positive correlated with a score between 0.1 and 0.29. The alternate hypotheses are: (i) H_{ar} that two resource counters are positive correlated with a score between 0.8 and 1, and (ii) H_{am} that two message types are positive correlated with a score between 0.8 and 1. We apply Fisher's z-transform and obtain the z-scores for all correlation coefficients [67]. When the absolute value of z at a confidence level of 99% is 2.64, it will reject the null hypothesis in favour of the alternate hypothesis.

Handling Inflation in False Positive

The technique we used to handle inflation in false positive due to testing multiple independent hypothesis is presented in Chapter 4.3.2 and we summarise the technique here. Our interest are on: (i) (strong) positive correlated resource use counters with a score between 0.8 and 1, and (ii) (strong) positive correlated message types with a score between 0.8 and 1. To test all the hypotheses, we apply a one-sided test. We use the significance level, $\alpha = 0.01$ to obtain an unadjusted P-value. The probability of identifying a significant correlation due to chance increases when more correlations are tested. The Bonferroni Correction [26] accounts for inflation in false positive by adjusting the P-value. To obtain the adjusted P-value, we multiply the unadjusted P-value by the number of dates.

Extracting the Nodes

Once the list containing the correlated significant messages and other messages is obtained, we implemented a process to extract the nodes associated with the correlated messages. The process is given as follows:

- Step 1: For each message type in the list of correlated messages, match the message type back to the system logs.

- Step 2: In the system logs, extract the log-entries that contain the message type to obtain a smaller set of system logs.
- Step 3: For all log-entries in the smaller set of system logs, extract the node and store it in a list of nodes.
- Step 4: Remove all repeated nodes in the list of nodes to obtain a list of unique nodes.

6.3 Case Studies on Ranger and Lonestar4

We conduct studies of rare error cases on the Ranger and Lonestar4 HPC systems. The Ranger HPC system was a 4,048 node Linux-based cluster and Lonestar4 was a 1,888 node Linux-based cluster. Both the Ranger and Lonestar4 HPC systems were operated by the Texas Advanced Computing Center at The University of Texas at Austin. On Ranger and Lonestar4, the TACC_Stats resource usage data were sampled at intervals of 10 minutes. On Ranger, we collected 26 days worth of resource use data and system logs. On Lonestar4, we collected 31 days worth of resource usage data and system logs. A summary of the data is given in Table 6.1.

Table 6.1: Summary of resource use data and system logs.

Ranger				
	TACC_Stats data		**Rationalized logs**	
No. days	Data size	No. of lines	Data size	No. of messages
26	124.1 GB	637,860,203	9.6 GB	64,822,682
Lonestar4				
	TACC_Stats data		**Syslogs**	
No. days	Data size	No. of lines	Data size	No. of messages
31	46.6 GB	207,068,692	1.3 GB	12,267,629

Therefore, given a cluster system with nodes, next we evaluate the PCA, ICA and NLPCA feature extractors and present four rare error cases.

6.3.1 Phase 1: Identify Significant Resource Use Counters and Messages

In this section, we determine on all dates: (i) the feature extractor which identifies the largest number of significant resource use counters, and (ii) the feature extractor which identifies the largest number of significant message types.

Identify Significant Resource Use Counters

Figure 6.2 shows the number of significant resource use counters that were identified by PCA, ICA and NLPCA on data matrices of 1 hour time-bins on Ranger.

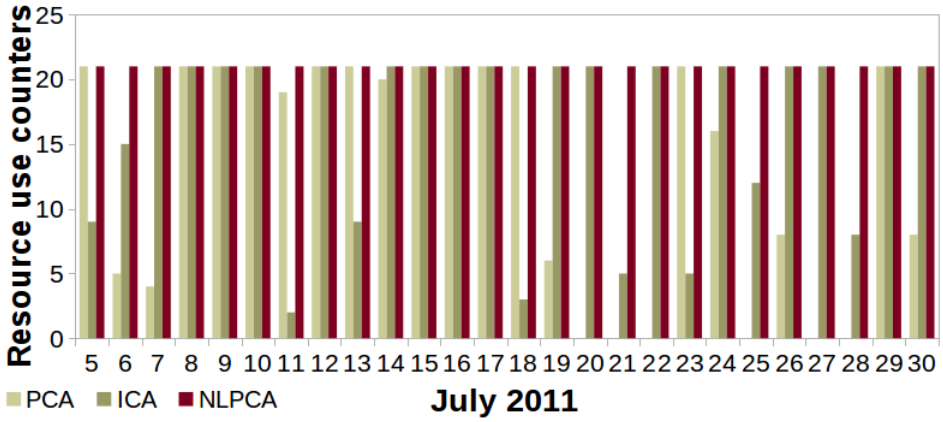


Figure 6.2: Significant resource use counters on time-bins of 1 hour on Ranger.

We observe that 21 resource use counters were identified by PCA on 12 dates, 21 resource use counters were identified by ICA on 17 dates and 21 resource use counters were identified by NLPCA on 26 dates.

Figure 6.3 shows the number of significant resource use counters that were identified on data matrices of 30 minute time-bins on Ranger.

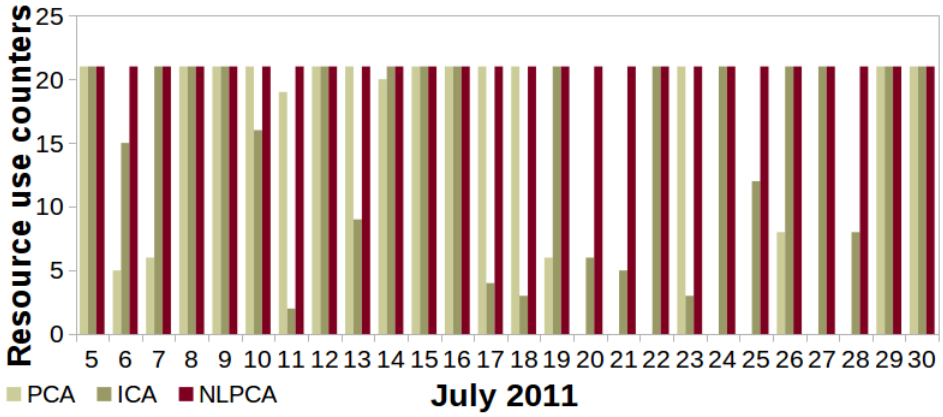


Figure 6.3: Significant resource use counters on time-bins of 30 minutes on Ranger.

We observe that 21 resource use counters were identified by PCA on 13 dates, 21 resource use counters were identified by ICA on 15 dates and 21 resource use counters were identified by NLPCA on 26 dates.

Figure 6.4 shows the number of significant resource use counters that were identified on data matrices of 10 minute time-bins on Ranger.

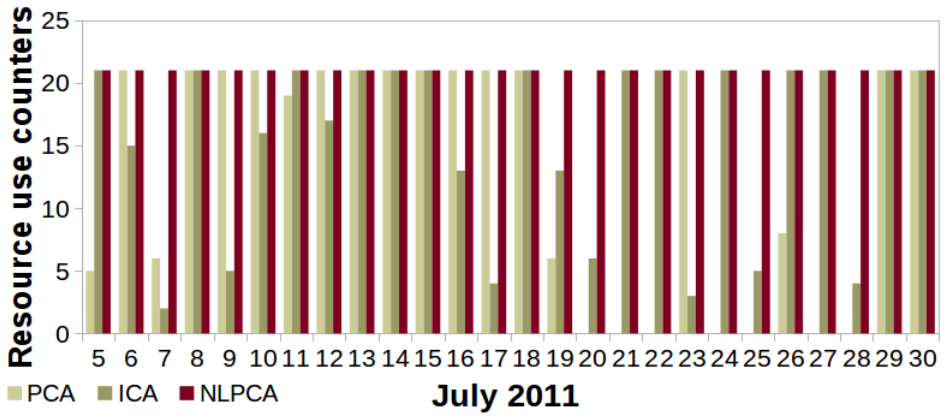


Figure 6.4: Significant resource use counters on time-bins of 10 minutes on Ranger.

We observe that 21 resource use counters were identified by PCA on 14 dates, 21 resource use counters were identified by ICA on 14 dates, and 21 resource use counters were identified by NLPCA on 26 dates. On the data matrices of 1 hour, 30 minute and 10 minute time-bins, both PCA and ICA took less than one second to execute. On the data matrices of 1 hour, 30 minute and 10 minute time-bins, NLPCA took an average of 55 seconds to execute. Our results show that:

- The largest number of significant resource use counters were identified only by NLPCA on all 26 dates in July 2011.
- The number of significant resource use counters that were identified on all the 26 dates represent 6% of all the resource use counters monitored on Ranger.

Figure 6.5 shows the number of significant resource use counters that were identified on data matrices of 1 hour time-bins on Lonestar4.

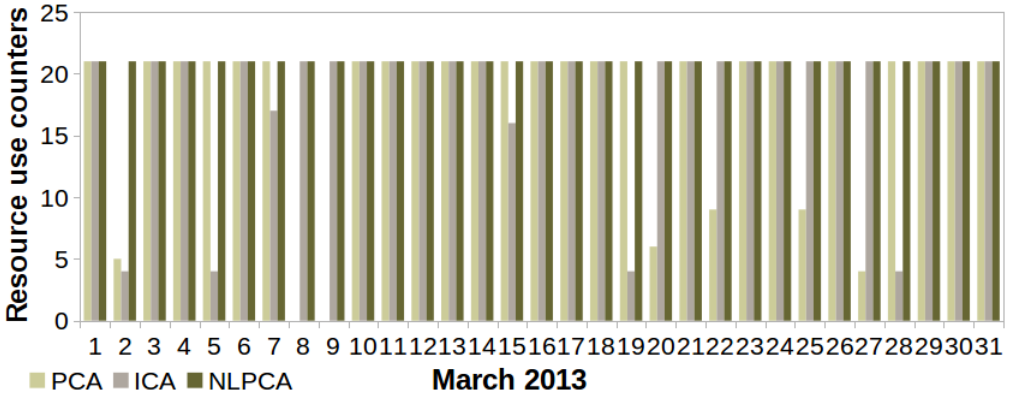


Figure 6.5: Significant resource use counters on time-bins of 1 hour on Lonestar4.

We observe that 21 resource use counters were identified by PCA on 24 dates,

21 resource use counters were identified by ICA on 25 dates, and 21 resource use counters were identified by NLPCA on 31 dates.

Figure 6.6 shows the number of significant resource use counters that were identified on data matrices of 30 minute time-bins on Lonestar4.

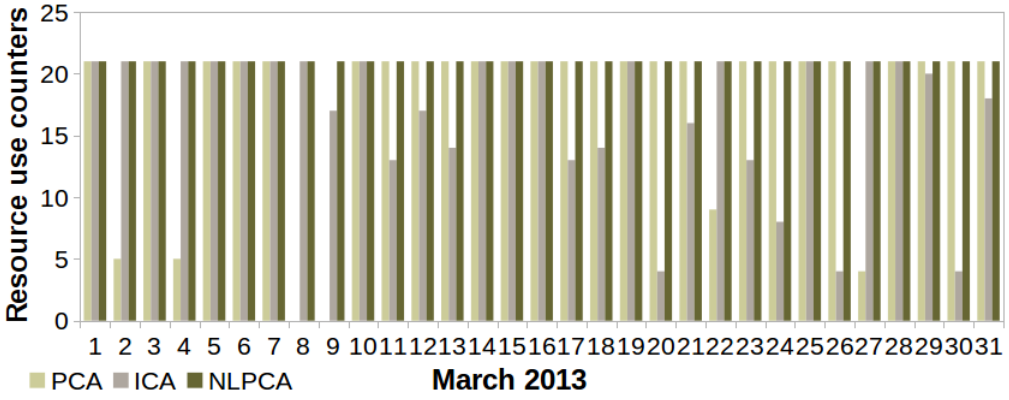


Figure 6.6: Significant resource use counters on time-bins of 30 minutes on Lonestar4.

We observe that 21 resource use counters were identified by PCA on 25 dates, 21 resource use counters were identified by ICA on 17 dates, and 21 resource use counters were identified by NLPCA on 31 dates.

Figure 6.7 shows the number of significant resource use counters that were identified on data matrices of 10 minute time-bins on Lonestar4.

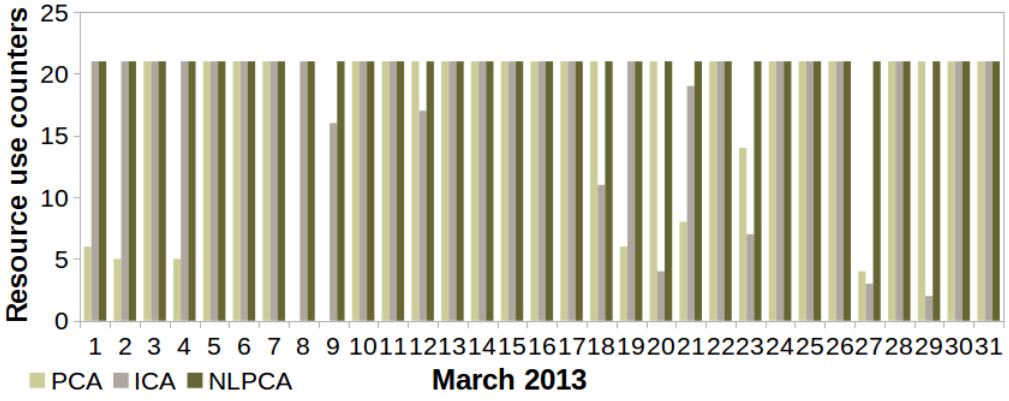


Figure 6.7: Significant resource use counters on time-bins of 10 minutes on Lonestar4.

We observe that 21 resource use counters were identified by PCA on 22 dates, 21 resource use counters were identified by ICA on 23 dates, and 21 resource use counters were identified by NLPCA on 31 dates. On the data matrices of 1 hour, 30 minute and 10 minute time-bins, both PCA and ICA took less than one second to execute. On the data matrices of 1 hour, 30 minute and 10 minute time-bins,

NLPCA took an average of 1 minute 22 seconds to execute. Our results show that:

- The largest number of significant resource use counters were identified only by NLPCA on all 31 dates in March 2013.
- The number of significant resource use counters that were identified on all the 31 dates represent 6% of all the resource use counters monitored on Lonestar4.

Identify Significant Message Types

Figure 6.8 shows the number of significant message types that were identified on data matrices of 1 hour time-bins on Ranger.

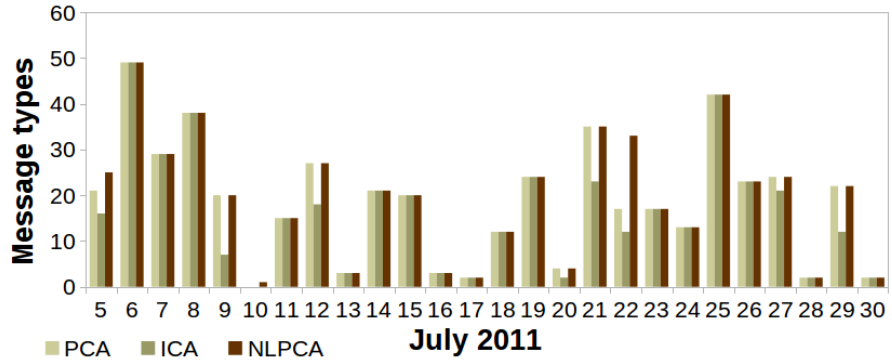


Figure 6.8: Significant message types on time-bins of 1 hour on Ranger.

We observe that: (i) the largest number of significant message types were identified by PCA on 23 dates, (ii) the largest number of significant message types were identified by ICA on 19 dates, and (iii) the largest number of significant message types were identified by NLPCA on 26 dates.

Figure 6.9 shows the number of significant message types that were identified on data matrices of 30 minute time-bins on Ranger.

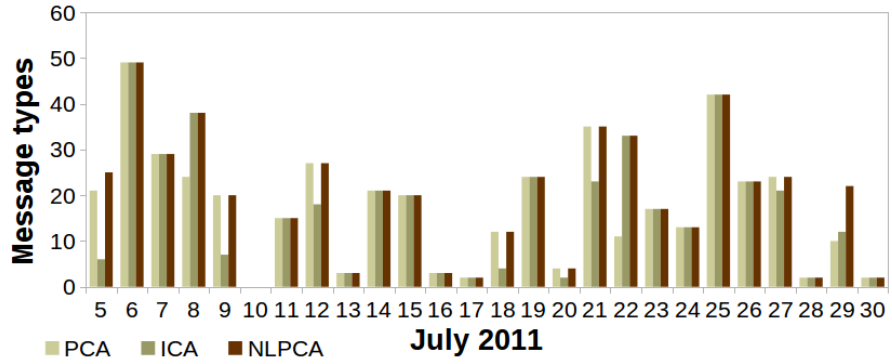


Figure 6.9: Significant message types on time-bins of 30 minutes on Ranger.

We observe that: (i) the largest number of significant message types were identified by PCA on 21 dates, (ii) the largest number of significant message types were identified by ICA on 17 dates, and (iii) the largest number of significant message types were identified by NLPCA on 25 dates.

Figure 6.10 shows the number of significant message types that were identified on data matrices of 10 minute time-bins on Ranger.

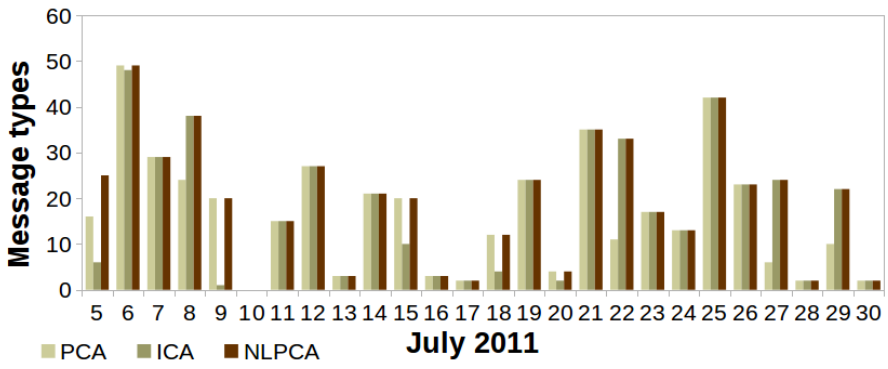


Figure 6.10: Significant message types on time-bins of 10 minutes on Ranger.

We observe that: (i) the largest number of significant message types were identified by PCA on 20 dates, (ii) the largest number of significant message types were identified by ICA on 19 dates, and (iii) the largest number of significant message types were identified by NLPCA on 25 dates. On the data matrices of 1 hour, 30 minute and 10 minute time-bins, PCA took 1.2 seconds to execute and ICA took less than one second to execute. On the data matrices of 1 hour, 30 minute and 10 minute time-bins, NLPCA took an average of 77 seconds to execute. Our results show that:

- Only NLPCA identified the largest number of significant message types on data matrices of 1 hour time-bins on all 26 dates in July 2011.
- The number of significant message types identified on all 26 dates represent between 5% and 50% of all message types generated on all the dates.

Figure 6.11 shows the number of significant message types that were identified on data matrices of 1 hour time-bins on Lonestar4. We observe that: (i) the largest number of message types were identified by PCA on 26 dates, (ii) the largest number of message types were identified by ICA on 18 dates, and (iii) the largest number of message types were identified by NLPCA on 31 dates.

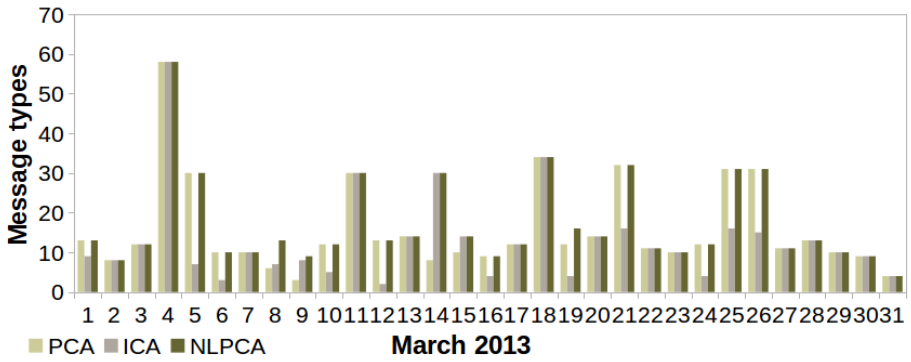


Figure 6.11: Significant message types on time-bins of 1 hour on Lonestar4.

Figure 6.12 shows the number of significant message types that were identified on data matrices of 30 minute time-bins on Lonestar4.

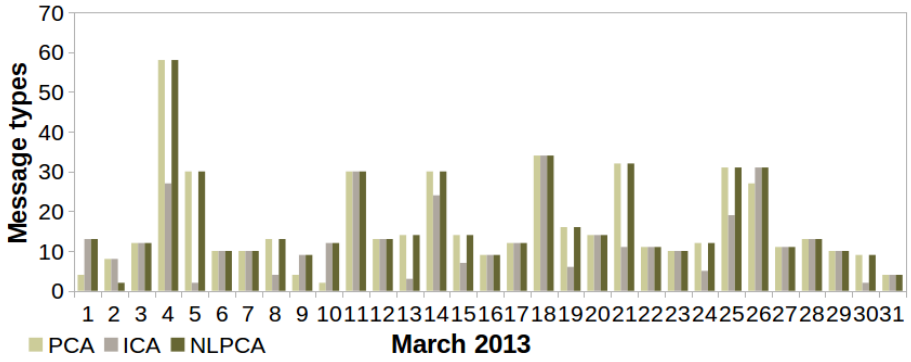


Figure 6.12: Significant message types on time-bins of 30 minutes on Lonestar4.

We observe that: (i) the largest number of message types were identified by PCA on 27 dates, (ii) the largest number of message types were identified by ICA on 20 dates, and (iii) the largest number of message types were identified by NLPCA on 30 dates.

Figure 6.13 shows the number of significant message types that were identified on data matrices of 10 minute time-bins on Lonestar4. We observe that: (i) the largest number of message types were identified by PCA on 28 dates, (ii) the largest number of message types were identified by ICA on 19 dates, and (iii) the largest number of message types were identified by NLPCA on 31 dates. On the data matrices of 1 hour, 30 minute and 10 minute time-bins, both PCA and ICA took 1.5 seconds to execute. On the data matrices of 1 hour, 30 minute and 10 minute time-bins, NLPCA took an average of 2 minutes 12 seconds to execute. Our results show that:

- Only NLPCA identified the largest number of significant message types on

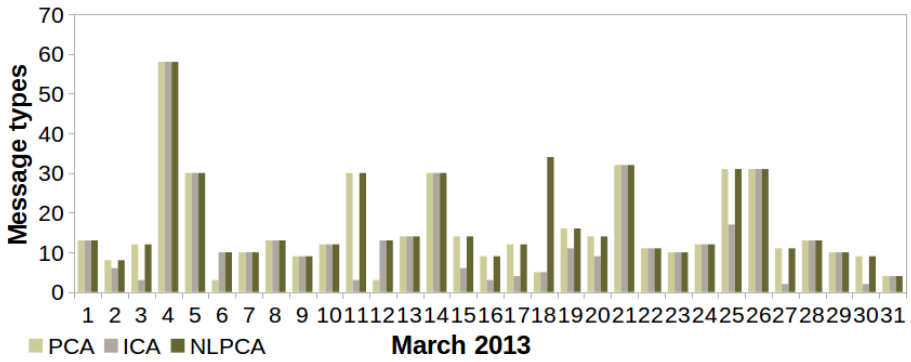


Figure 6.13: Significant message types on time-bins of 10 minutes on Lonestar4.

data matrices of 1 hour and 10 minute time-bins on all 31 dates in March 2013.

- The number of significant message types identified on all 31 dates represent between 3% and 46% of message types generated on all the dates.

6.3.2 Phase 2: Identify Rare Error Cases

In this section, we provide details for four error cases on the Ranger and Lonestar4 HPC systems. We use the lists of correlated resource use counters and correlated errors to identify the error cases and discussed the use cases with the system administrator. In all the error cases, the errors are different. The errors are: (i) network packet drops and Lustre I/O errors, (ii) CPU I/O bottleneck and Lustre client eviction, (iii) virtual memory and harddisk I/O errors, and (iv) data and network synchronisation errors. A summary of the error cases is given in Table 6.2.

Table 6.2: List of rare error cases on Ranger and Lonestar4.

Ranger		
Components	**Error**	**Date**
Infiniband & Lustre network, Lustre filesystem	Network packet drops & Lustre I/O	July 5
CPU, harddisk & Lustre filesystem	CPU I/O bottleneck & Lustre client eviction	July 15
Memory management, harddisk & Lustre	Virtual memory & harddisk I/O	July 25
Lonestar4		
Components	**Error**	**Date**
Memory, Linux threads & Lustre network	Data & network synchronisation	March 1

When the Linux operating system kernel goes into a loop, i.e., Linux hanged, a soft lockup event is generated. To identify the soft lockup event, we scan the

system logs for a message containing the keywords `soft lockup`. We implemented a function in EXERMEST to search the system logs for soft lockup events and extract the dates of soft lockups. In the Ranger system logs, we identified 12 dates of soft lockups for July 2011 (see Figure 3.2b). In the Lonestar4 system logs, we identified seven dates of soft lockups for March 2013 (see Figure 3.3b).

Error Case 1: Infiniband, Lustre Network and Lustre Filesystem

The network interface card (NIC) transmits and receives data packets on a network. When a NIC receives or transmits data, first it will store the data into the available buffers on the network card. Most of the time, the network card receives the data into its buffers correctly. However, in a rare case, the buffers on the network card may be full and some of the data is dropped. A networked filesystem transmits and receives a lot of data on the network. When a filesystem on a node wants to send data to another node, it will send the data to the network card first. Most of the time, the network card receives the data into its buffers correctly. However, if the data size is larger than the available buffers on the network card, the NIC is not able to transmit all the data.

We manually scan the lists of significant resource use counters generated by PCA, ICA and NLPCA on data matrices of 1 hour, 30 minute and 10 minute time-bins. We found that only the list generated on the resource use data matrix of 10 minute time-bins contain both `net ib0 tx_dropped` and `llite /work write_bytes`. Further, we identified the two resource use counters only on July 5 2011. We found that both the PCA and ICA generated lists contain the `llite /work write_bytes` resource use counter. Therefore, the Lustre filesystem write bytes resource use counter follows a linear uncorrelated pattern – in this case, we can use PCA as the primary method. We found that only the NLPCA generated list contain the `net ib0 tx_dropped` resource use counter. Our result shows that: (i) the network transmit packet drop resource use counter follows a non-linear uncorrelated pattern on July 5 2011, and (ii) PCA and NLPCA are required for identifying the significant `net ib0 tx_dropped` and `llite /work write_bytes` resource use counters.

Next, we scan the lists of correlated resource use counters generated on the data matrix of 10 minute time-bins. Figure 6.14 shows the significant resource use counters that are correlated to other resource use counters. We observe a strong correlation of `net ib0 tx_dropped` to `lnet rx_msgs_dropped`, `llite /work dirty_pages_misses` and `llite /work ioctl` – the correlation strength ranges from 0.93 to 1. We observe a strong correlation of `llite /work write_bytes` to `lnet rx_msgs_dropped`, `llite /work dirty_pages_misses` and `llite /work ioctl` – the correlation strength ranges from 0.93 to 1. We found that the correlated resource use coun-

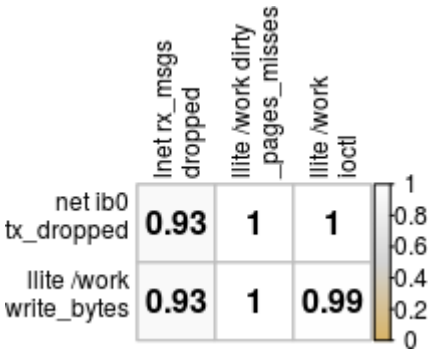


Figure 6.14: Correlated resource use counters on Ranger. The significant resource use counters are "net ib0 tx_dropped" and "llite /work write_bytes".

ters are contained only in the list generated by Pearson correlation. Pearson correlation took 1.5 seconds to execute and generate the list. The `lnet rx_msgs_dropped` records dropped packets on the Lustre filesystem network. Our result shows that Pearson correlation is a suitable method; it identified the Lustre I/O, Infiniband and Lustre network resource use counters that follow a linear pattern.

The `IO window` and `PREFETCH window` events can be used to see what happens when the Linux kernel executes a function call. We manually scan the lists of significant message types generated by PCA, ICA and NLPCA on the 1-hour, 30 and 10 minutes time-bins data matrices. We found that only the lists generated on the 1-hour time-bins message types data matrix contain the `IO window` and `PREFETCH window` events. Further, we identified the two system events only on July 5. We found both the system events only in the NLPCA generated list. Our result shows that the `IO window` and `PREFETCH window` events follow a non-linear uncorrelated pattern on July 5 2011.

Next, we scan the lists of correlated messages that were generated on the 1-hour time-bins data matrix on July 5 2011. Figure 6.15 shows the significant system events that are correlated to other system events. We observe a strong correlation of `IO window` to `error reading dir`, `request timed out` and `tried all connections ...` – the correlation strength ranges from 0.88 to 1. We observe a strong correlation of `PREFETCH window` to `error reading dir`, `request timed out` and `tried all connections ...` – the correlation strength ranges from 0.88 to 1. We found that only the list generated by Pearson correlation contain the correlated system events. Our result shows that Pearson correlation is a suitable method; it identified that the system events follow a linear pattern on July 5 2011.

Correlation with failures: Next, we scan the correlation matrix that was generated on the 1-hour time-bins data matrices to determine the correlation strength

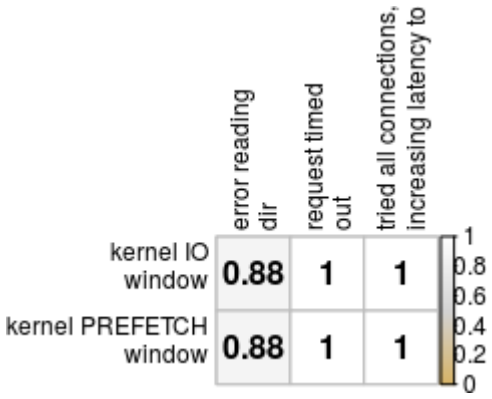


Figure 6.15: Correlated messages on Ranger. The significant events are "kernel IO window" and "kernel PREFETCH window".

of `IO window` and `PREFETCH window` to `soft lockup` events. We found that their correlation scores is -0.07. Therefore, there is no correlation between the two Linux function call messages and soft lockup events on July 5 2011.

Detailed diagnosis: On July 5 2011, data that was transmitted was dropped by a NIC due to insufficient buffer space on the network card. On the same day, a filesystem directory read error was strongly correlated to two Linux kernel function calls. Further, two timeout and reconnection messages were strongly correlated to both Linux kernel function calls. A soft lockup was reported on July 5 2011 but there is no correlation of both the kernel function calls to the soft lockup event. Therefore, the networked filesystem error did not cause compute node crash on July 5 2011.

When on the same day, correlations of network transmit packet drop, Lustre I/O and Lustre network resource use counters and correlations of Linux I/O, Lustre I/O and network error messages occur, we can use the network transmit packet drop and Lustre I/O counters to monitor the state of the Infiniband and Lustre network and filesystem.

Validation: We test the significance of: (i) strong positive correlated resource use counters, and (ii) strong positive correlated messages. We summarise their z-scores in Table 6.3. All the z-scores range from 4.85 to 27.73. At the confidence level of 99%, under the null hypothesis $z_{0r} = 2.64$ and $z_{0m} = 2.64$. Therefore, we reject the null hypothesis in favour of the alternate hypothesis.

Table 6.3: z-scores for correlated resource use counters and correlated messages on Ranger.

Correlated resource use counters	**July 5 2011**
Infiniband network, Lustre network & I/O	$16.00 \leq z_r \leq 27.73$
Correlated messages	**July 5 2011**
Kernel function call, Lustre I/O & timeout errors	$4.85 \leq z_m \leq 10.68$

Next, we determine the probability of identifying a significant result due to the increase in the number of hypotheses tested. The lowest z-score in Table 6.3 is 4.85. We apply a one-sided test and use the significance level, $\alpha = 0.01$ for all given hypothesis tests to obtain a P-value. The P-value is equal to $P(Z > 4.85) = 1 - P(Z \leq 4.85) = 1 - 0.9999 = 0.0001$. Then, we obtain the adjusted P-value $0.0001 \times 26 = 0.0026$ where the number of dates is 26. For all z-scores in Table 6.3, the adjusted P-values are less than 0.01. This indicates that all the correlations would not be observed under the null hypothesis.

Identify the nodes: We determine the nodes which are associated with the identified errors. A summary is given in Table 6.4.

Table 6.4: List of messages and associated number of nodes on Ranger.

Significant message	**Nodes**	**Correlated message**	**Nodes**
IO window	6	error reading dir	1
PREFETCH window	6	tried all connections,	3,929
		request timed out	3,929

From Table 6.4, we observe:

- there are six nodes associated with `IO window` and `PREFETCH window`.
- there is one node associated with `error reading dir`.
- there are 3,929 nodes associated with `request timed out` and `tried all connections`.

Error Case 2: CPU, Harddisk and Lustre Filesystem

When data is prepared to be written to the harddisk, the filesystem sends a write request to the harddisk. The resource use counter named `block hda wr_merges` is incremented when a write request merges with the existing in-queue requests. When the filesystem performs an I/O operation, the processor waits for the I/O operation to complete. The resource use counter named `cpu 6 idle` is incremented when a processor waits on a filesystem I/O operation.

We manually scan the lists of significant counters generated by PCA, ICA and NLPCA on the data matrices of 1 hour, 30 minute and 10 minute time-bins. We found that only the lists generated on the 10 minutes time-bins data matrix contain `cpu 6 iowait` and `llite /work ioctl`. Further, we identified both the resource use counters only in the list on July 15. We found both the resource use counters only in the list generated by NLPCA. Our results show that the CPU I/O wait and Lustre filesystem I/O resource use counters follow a non-linear uncorrelated pattern on July 15 2011.

Next, we scan the lists of correlated resource use counters generated on the 10 minutes time-bins data matrices. Figure 6.16 shows the significant resource use counters that are correlated to other resource use counters.

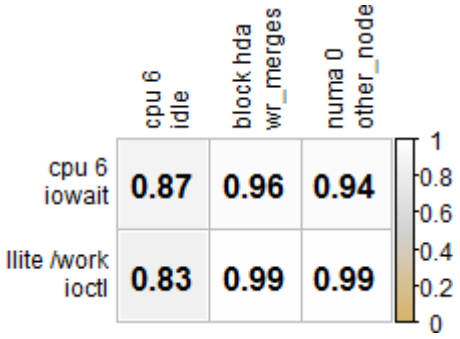


Figure 6.16: Correlated resource use counters on Ranger. The significant resource use counters are "cpu iowait" and "llite /work ioctl".

We observe there is a strong correlation of `cpu 6 iowait` to `cpu 6 idle`, `block hda wr_merges` and `numa 0 other_node` – the correlation strength ranges from 0.87 to 0.96. We observe there is a strong correlation of `llite /work ioctl` to `cpu 6 idle`, `block hda wr_ merges` and `numa 0 other_node` – the correlation strength ranges from 0.83 to 0.99. We found the correlated resource use counters in the lists generated by Pearson and Spearman-Rank correlation methods. The Pearson and Spearman-Rank correlation methods took 1.2 seconds to execute and generate the lists. Our result shows that Spearman-Rank and Pearson correlation are suitable methods. When both correlation methods identify the correlated events, Pearson correlation can be used as the primary method.

The `connection lost` and `connection restored` events can be used to see what happens when the Lustre filesystem communicates with an unresponsive client. We scan the lists of significant message types generated by PCA, ICA and NLPCA on the 1-hour, 30 and 10 minutes time-bins data matrices. We found that only the lists generated on the 10 minutes time-bins data matrix contain `connection lost` and `connection restored` events. Further, we identified the two events only in the lists on July 15. We found the two events only in the list generated by ICA. Our result shows that the two system events follow a linear uncorrelated pattern on July 15 2011.

Next, we scan the lists of correlated messages generated on the 10 minutes time-bins data matrix. Figure 6.17 shows the significant events that are correlated to other system events. We observe there is a strong correlation of `connection was lost` to `cancel RPC`, `request timed out` and `client was evicted` – the correlation strength is 0.86. We observe there is a strong correlation of `connection restored`

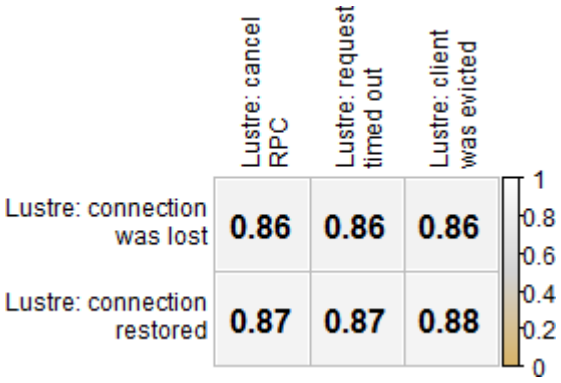


Figure 6.17: Correlated messages on Ranger. The significant events are "connection was lost" and "connection restored".

to `cancel RPC`, `request timed out` and `client was evicted` – the correlation strength ranges from 0.87 to 0.88. We found the correlated events only in the list generated by Pearson correlation. Our results show that the system events follow a linear pattern on July 15 2011.

Correlation with failures: No soft lockup event was reported on July 15 2011 (see Figure 3.2b for the dates of compute node soft lockups on July 2011.). Therefore, the Lustre filesystem eviction of an unresponsive client did not cause a compute node to crash.

Detailed diagnosis: On July 15 2011, the Lustre filesystem was executing I/O operation which caused the processor to wait for the I/O operation to finish. On the same day, eviction of an unresponsive Lustre client was traced to three correlated Lustre Evict/RPC protocol messages. On the same day, a NUMA resource use counter which records the number of memory pages allocated to a remote process, is correlated with CPU idle and harddisk I/O activity counters.

When on the same day: (i) correlations of significant cpu iowait activity to cpu idle and harddisk I/O activities, and correlations of significant Lustre I/O activity to cpu idle and harddisk I/O activities occur, and (ii) correlations of Lustre connection and client eviction error events also occur it shows that Lustre network and client evicted errors are generated by CPU, harddisk and Lustre filesystem activities. Therefore, we can use the CPU I/O wait and Lustre filesystem I/O resource use counters to monitor the state of the CPU, harddisk and Lustre filesystem.

Validation: We test the significance of: (i) strong positive correlated resource use counters, and (ii) strong positive correlated messages. We summarise their z-scores in Table 6.5. All the z-scores range from 10.42 to 27.73. At the confidence level of 99%, under the null hypothesis $z_{0r} = 2.64$ and $z_{0m} = 2.64$. Therefore, we reject the null hypothesis in favour of the alternate hypothesis.

Table 6.5: z-scores for correlated resource use counters and messages on Ranger.

Correlated resource use counters	**July 15 2011**
CPU, harddisk, NUMA & Lustre I/O	$10.42 \leq z_r \leq 27.73$
Correlated messages	**July 15 2011**
Unresponsive Lustre client & Evict/RPC	$11.67 \leq z_m \leq 12.64$

Next, we determine the probability of identifying a significant result due to the increase in the number of hypotheses tested. We apply a one-sided test and use the significance level, $\alpha = 0.01$ for all given hypothesis tests to obtain a P-value. From Table 6.5, we observe that the lowest z-score is 10.42. Since this is a one-sided test, the P-value is equal to $P(Z > 10.42) = 1 - P(Z \leq 10.42) = 1 - 0.99999 = 0.00001$. Then, we obtain the adjusted P-value $0.00001 \times 26 = 0.00026$ where $d = 26$. For all z-scores in Table 6.5, the adjusted P-values are less than 0.01. This indicates that it is highly unlikely that all the correlations would be observed under the null hypothesis.

Identify the nodes: We determine the nodes which are associated with the identified errors. A summary is given in Table 6.6.

Table 6.6: List of messages and associated number of nodes on Ranger.

Significant message	**Nodes**	**Correlated message**	**Nodes**
connection restored	152	cancel RPC	12
connection was lost	152	client was evicted	12
		request timed out	151

From Table 6.6, we observe:

- there are 152 nodes associated with `connection restored` and `connection was lost`.
- there are 12 nodes associated with `cancel RPC` and `client was evicted`.
- there are 151 nodes associated with `request timed out`.

Error Case 3: Linux Memory Management Unit, Harddisk and Lustre Filesystem

When a program makes a request for data, the operating system checks that the data requested by the program is available in main memory. If the data is not in the main memory and the main memory is full, some existing data is moved to the harddisk to free up space for new data. The resource use counter named `mem Writeback` records the total amount of data that is moved from the memory to harddisk. The resource

use counter named `block hda wr_sectors` records the total number of harddisk sectors that were written.

We manually scan the lists of significant resource use counters generated by PCA, ICA and NLPCA on the 1-hour, 30 and 10 minutes time-bins data matrices. We found that: (i) only the lists generated on the 10 minutes time-bins data matrix contain `vm pgfault` and `llite /share ioctl`, (ii) only the list generated by ICA contain `vm pgfault`, and (iii) only the list generated by NLPCA contain `llite /share ioctl`. Further, we identified the two resource use counters only in the lists on July 25. The page-fault resource use counter follows a linear uncorrelated pattern. The Lustre filesystem I/O resource use counter follows a non-linear uncorrelated pattern. Our result shows that ICA and NLPCA are required for identifying the two significant resource use counters on July 25 2011.

Next, we scan the lists of correlated resource use counters generated on the 10 minutes time-bins data matrices. Figure 6.18 shows the significant resource use counters that are correlated to other resource use counters.

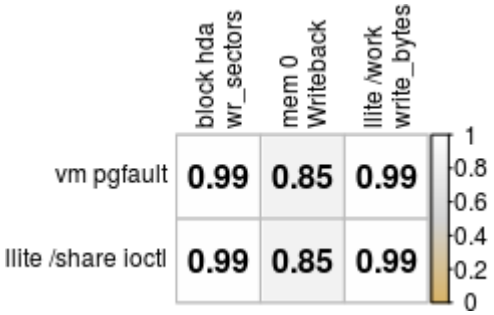


Figure 6.18: Correlated resource use counters on Ranger. The significant resource use counters are "vm pgfault" and "llite /share ioctl".

We observe there is a strong correlation of `vm pgfault` to `block wr_sectors`, `mem Writeback` and `llite /work write_bytes` – the correlation scores range from 0.85 to 0.99. We observe there is a strong correlation of `llite /share ioctl` to `block wr_sectors`, `mem Writeback` and `llite /work write_bytes` – the correlation scores range from 0.85 to 0.99. We found that both the Pearson and Spearman-Rank correlation generated lists contain the correlated resource use counters. The Pearson and Spearman-Rank correlation methods took 1.5 seconds to execute and generate the lists of correlated counters. Our result shows that Spearman-Rank and Pearson correlation are suitable methods. When the correlated events are identified by both correlation methods, Pearson correlation can be used as the primary method.

The `get_user_pages` and `copy_strings` system events can be used to see what happens when Linux's memory management unit functions are executed. We scan the lists of significant message types generated by PCA, ICA and NLPCA

on the 1-hour, 30 and 10 minutes time-bins data matrices. We found that only the lists generated on the 1-hour time-bins message types data matrix contain the `get_user_pages` and `copy_strings` events. Further, we identified the two system events only in the list on July 25. We found the two system events only in the list generated by NLPCA. Our result shows that NLPCA identified that the two system events follow a non-linear uncorrelated pattern on July 25 2011.

Next, we scan the lists of correlated messages generated on the 1-hour time-bins data matrix. Figure 6.19 shows the significant system events that are correlated to other system events.

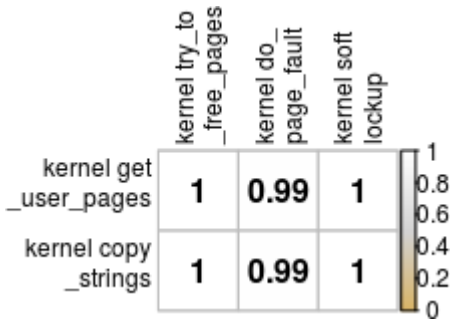


Figure 6.19: Correlated messages on Ranger. The significant events are "kernel get_user_pages" and "kernel copy_strings".

We observe there is a strong correlation of `get_user_pages` to `try_to_free_pages`, `do_page_fault` and `soft lockup` – the correlation score is 0.99. We observe there is a strong correlation of `copy_strings` to `try_to_free_pages`, `do_page_fault` and `soft lockup` – the correlation score is 0.99. We found that the correlated system events are present only in the list generated by Pearson correlation. Our results show that the correlated system events follow a linear pattern on July 25 2011.

Correlation with failures: From Figure 6.19, we observe that there is a strong correlation of `get_user_pages` and `copy_strings` to `soft lockup` – the correlation score is 1.

Detailed diagnosis: On July 25 2011, two kernel functions were executed by the Linux memory management unit to free up available space in the system main memory. The decision to free up space in main memory was made in response to a program's request for data. A page-fault occurred which led to a Linux operating system crash.

When on the same day: (i) correlations of significant page-fault activity to memory, harddisk I/O and filesystem activities, and correlations of significant Lustre I/O activity to memory, harddisk I/O and filesystem activities occur, and (ii) correlations of Linux memory management, page-fault error events and soft lockup messages also occur, it shows that virtual memory and harddisk I/O errors are

generated by page-fault, filesystem, harddisk and memory I/O activities. Therefore, we can use the page-fault and Lustre filesystem I/O resource use counters to monitor the state of Linux memory management.

Validation: We test the significance of: (i) strong positive correlated resource use counters, and (ii) strong positive correlated messages. We summarise their z-scores in Table 6.7. All the z-scores range from 10.68 to 27.73. At the confidence level of 99%, under the null hypothesis $z_{0r} = 2.64$ and $z_{0m} = 2.64$. Therefore, we reject the null hypothesis in favour of the alternate hypothesis.

Table 6.7: z-scores for correlated resource use counters and messages on Ranger.

Correlated resource use counters	**July 25 2011**
Page-fault, hard disk, memory & Lustre I/O	$11.22 \leq z_r \leq 27.73$
Correlated messages	**July 25 2011**
Memory management, page-fault & Linux crash	$z_m = 10.68$

Next, we determine the probability of identifying a significant result due to the increase in the number of hypotheses tested. We apply a one-sided test and use the significance level, $\alpha = 0.01$ for all given hypothesis tests to obtain a P-value. From Table 6.7, we observe that the lowest z-score is 10.68. Since this is a one-sided test, the P-value is equal to $P(Z > 10.68) = 1 - P(Z \leq 10.68) = 1 - 0.99999 = 0.00001$. Then, we obtain the adjusted P-value $0.00001 \times 26 = 0.00026$ where $d = 26$. For all z-scores in Table 6.7, the adjusted P-values are less than 0.01. This indicates that it is highly unlikely that all the correlations would be observed under the null hypothesis.

Identify the nodes: We determine the nodes which are associated with the identified errors. A summary is given in Table 6.8.

Table 6.8: List of messages and associated nodes on Ranger.

Significant message	**Node**	**Correlated message**	**Node**
get_user_pages	i182-312	do_page_fault	i128-406, i182-312
copy_strings	i182-312	try_to_free_pages	i182-312
		soft lockup	i128-406, i162-208, i182-312

From Table 6.8, we observe:

- there is one node associated with `get_user_pages` and `copy_strings`.
- there are two nodes associated with `do_page_fault`.
- there is one node associated with `try_to_free_pages`.
- there are three nodes associated with `soft lockup`.

Error Case 4: Dirty Memory, Linux Threads and Lustre Network

When data in the main memory is modified but not yet written back to the harddisk, the in-memory version of the data is out of sync with the version on disk. The resource use counter named `mem Dirty` records the amount of data that is out of sync. When the version on the harddisk is updated, the data in main memory and harddisk are in sync. The resource use counter named `mem Writeback` records the amount of data that is actively being written to the disk. Data in memory must be synchronised with the data on disk when multiple threads belonging to a process are created. In most cases, the data in memory is synchronised with the data on disk. However, on a rare occasion, if one of the threads fail then some of the data in memory and disk may not be synchronised. The resource use counter named `ps nr_threads` records the number of threads.

We scan the list of significant resource use counters generated by PCA, ICA and NLPCA on the 10 and 30 minutes and 1 hour time-bins data matrices. We found that: (i) the resource use counter `ps nr_threads` is contained only in the list generated by NLPCA on the 10 minutes time-bins data matrix, and (ii) the resource use counter `mem 0 Dirty` is contained in the lists generated by NLPCA on the 10 minutes time-bins data matrix and ICA on the 1 hour time-bins data matrix. Further, `ps nr_threads` and `mem 0 Dirty` are contained only in the lists of significant resource use counters on March 01 2013. Our results show that:

- the resource use counter `ps nr_threads` follows a non-linear uncorrelated pattern on March 01 2013. Therefore, NLPCA is a suitable method for identifying `ps nr_threads`.

- the resource use counter `mem 0 Dirty` follows: (i) a non-linear uncorrelated pattern on 10 minutes time-bins, and (ii) a linear uncorrelated pattern on 1 hour time-bins on March 01 2013. Because ICA generates the list of significant resource use counters faster than NLPCA, when NLPCA and ICA identifies the significant resource use counter, ICA can be used as the primary method.

Next, we scan the lists of correlated resource use counters that were generated on the 10 minutes time-bins and 1 hour time-bins data matrices. Figure 6.20 shows the significant resource use counters that are correlated to other resource use counters. We observe there is a strong correlation of `ps nr_threads` to `mem 1 Writeback`, `mem 1 Dirty` and `mem 0 LowFree` – the correlation scores range from 0.81 to 0.82. We observe there is a strong correlation of `mem 0 Dirty` to `mem 1 Writeback`, `mem 1 Dirty` and `mem 0 LowFree` – the correlation scores range from 0.8 to 0.95. We found that only Spearman-Rank correlation identified the correlated `ps nr_threads` and `mem 1 Writeback`, `ps nr_threads` and `mem 1 Dirty`,

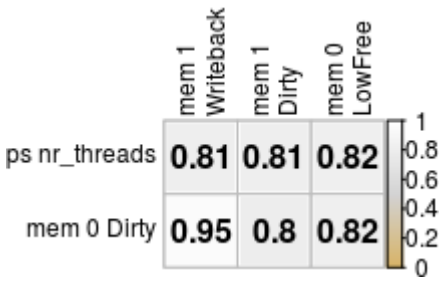


Figure 6.20: Correlated resource use counters on Lonestar4. The significant resource use counters are "ps nr_threads" and "mem 0 Dirty".

`mem 0 Dirty` and `mem 1 Dirty`, and `mem 0 Dirty` and `mem 0 LowFree`. We found that only Pearson correlation identified the correlated `mem 0 Dirty` and `mem 1 Writeback`. The correlated `ps nr_threads` and `mem 0 LowFree` resource use counters are contained in the lists generated by Pearson and Spearman-Rank correlation methods. Both Pearson and Spearman-Rank correlation methods took 1.5 seconds to generate the lists. Our results show that both Pearson and Spearman-Rank correlation are required for identifying the correlated resource use counters.

The system events `failed to close new saved state file` and `request sent has timed out` can be used to see what happens when synchronisation errors occur. We scan the lists of significant messages generated by PCA, ICA and NLPCA on the 10 and 30 minutes and 1 hour time-bins data matrices. We found that: (i) the event `failed to close new saved state file` is contained only in the list generated by NLPCA on the 30 minutes time-bins data matrix, and (ii) the event `request sent has timed out` is contained only in the lists generated by ICA on the 10 minutes and 1 hour time-bins data matrices. Further, the two error events are contained only in the lists of significant messages on March 01 2013. Our results show that:

- the event `failed to close new saved state file` follows a non-linear uncorrelated pattern on March 01 2013. Therefore, NLPCA is a suitable method for identifying the event.

- the event `request sent has timed out` follows a linear uncorrelated pattern on March 01 2013. Therefore, ICA is a suitable method for identifying the event.

Next, we scan the lists of correlated messages generated on the 10 and 30 minutes time-bins data matrices. Figure 6.21 shows the significant messages that are correlated to other messages. We observe there is a strong correlation of `failed to close new saved state file` to `data not saved properly` – the

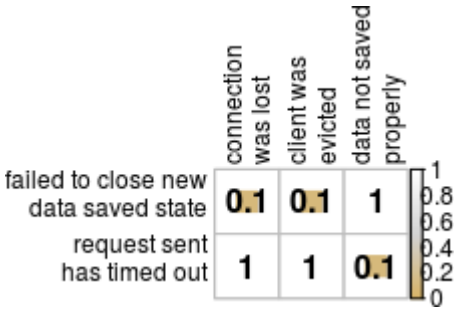


Figure 6.21: Correlated messages on Lonestar4. The significant messages are "failed to close new data saved state" and "request sent has timed out".

correlation score is 1. We observe there is a strong correlation of `request sent has timed out` to `connection was lost` and `client was evicted` – the correlation score is 1. We found that the correlated messages are contained in the lists generated by Pearson and Spearman-Rank correlation methods. Both Pearson and Spearman-Rank correlation took 1.5 seconds to generate the list of correlated messages. When both Pearson and Spearman-Rank correlation methods identify the correlated messages, Pearson correlation can be used as the primary method.

Correlation with failures: There are no soft lockup events on March 01 2013.

Detailed diagnosis: On March 01 2013, two synchronisation error events were reported by the Linux operating system. The decision to synchronise data in memory and disk was identified from the strong correlations of Linux process threads to dirty memory and memory writeback resource use counters. The synchronisation of data and the generation of synchronisation error events did not lead to a compute node crash on March 01 2013.

When on the same day: (i) correlations of significant Linux process threads and dirty memory to memory writeback activities occur, and (ii) correlations of significant network and data saved errors to Lustre network and data saved events also occur, it shows that data and network synchronisation errors are generated by Linux process threads and Lustre network events. Therefore, we can use the Linux process threads and dirty memory resource use counters and Lustre network events to monitor the state of data and network synchronisation.

Validation: We test the significance of: (i) strong positive correlated resource use counters, and (ii) strong positive correlated messages. We summarise their z-scores in Table 6.9. All the z-scores range from 6.98 to 27.73. At the confidence level of 99%, under the null hypothesis $z_{0r} = 2.64$ and $z_{0m} = 2.64$. Therefore, we reject the null hypothesis in favour of the alternate hypothesis.

Next, we determine the probability of identifying a significant result due to the increase in the number of hypotheses tested. We apply a one-sided test and use

Table 6.9: z-scores for correlated resource use counters and messages on Lonestar4.

Correlated resource use counters	**March 01 2013**
Linux threads & memory	$6.98 \leq z_r \leq 9.36$
Correlated messages	**March 01 2013**
Data synchronisation & Lustre network errors	$23.41 \leq z_m \leq 27.73$

the significance level, $\alpha = 0.01$ for all given hypothesis tests to obtain a P-value. From Table 6.9, we observe that the lowest z-score is 6.98. Since this is a one-sided test, the P-value is equal to $P(Z > 6.98) = 1 - P(Z \leq 6.98) = 1 - 0.99997 = 0.00003$. Then, we obtain the adjusted P-value $0.00003 \times 31 = 0.00093$ where $d = 31$. For all z-scores in Table 6.9, the adjusted P-values are less than 0.01. This indicates that it is highly unlikely that all the correlations would be observed under the null hypothesis.

Identify the nodes: We determine the nodes which are associated with the identified errors. A summary is given in Table 6.10.

Table 6.10: List of messages and associated nodes on Lonestar4.

Significant message	**Node**	**Correlated message**	**Node**
failed to close new saved state file	login3	data not saved properly	login3
request sent has timed out	15	connection was lost	17
		This client was evicted	17

From Table 6.10, we observe:

- there is one node associated with `failed to close new saved state file` and `data not saved properly`.
- there are 15 nodes associated with `request sent has timed out`.
- there are 17 nodes associated with `connection was lost` and `This client was evicted`.

6.4 Summary

In this chapter, we presented the EXERMEST framework for diagnosing rare error cases on HPC systems. The main technical contribution is a new workflow that integrated three feature extraction methods and two correlation methods. We evaluated the feature extractors and combined resource use data matrices with message types data matrices to identify the significant resource use counters and system events associated with rare error cases. EXERMEST identified four error cases on two HPC systems and diagnosed multiple system components associated

with the errors. We showed that multiple feature extractors and time-bins of different granularities are required for identifying the error cases. EXERMEST used the Bonferroni correction to ensure the accuracy of the diagnoses and showed that all the correlations would not be observed under the null hypothesis.

Chapter 7

A Comparative Analysis of CORRMEXT and EXERMEST

In this chapter, we discuss the similarities and differences between the CORRMEXT (**COR**relating **R**esource use data and **M**essage logs and **EX**tracting **T**imes) and EXERMEST (**EX**tracting F**E**atures and Co**R**relating Resource Use Counters and **MES**sage **T**ypes) diagnostics frameworks. Whereas the CORRMEXT framework combines data type extraction, multiple correlation methods and time-bin extraction to identify frequently occurring error cases and report the success and failure of error recovery protocols, the EXERMEST framework combines multiple feature extraction methods and multiple correlation methods to identify rare error cases. We provide this chapter for the interested reader to easily compare the diagnostics capabilities of CORRMEXT and EXERMEST.

We structure this chapter as follows: In Section 7.1, we summarise the CORRMEXT framework and describe the list of error cases. In Section 7.2, we summarise the EXERMEST framework and describe the list of error cases. In Section 7.3, we discuss the similarities and differences between CORRMEXT and EXERMEST and conclude with a summary in Section 7.4.

7.1 CORRMEXT Failure Diagnosis Framework

In this section, we summarise the CORRMEXT diagnostics framework and describe the list of error cases.

7.1.1 Introduction

We developed CORRMEXT to study patterns of error cases that occur frequently. CORRMEXT used real resource use data and system logs in its analyses. The

resource use data and system logs were generated on production high performance computing systems. CORRMEXT combined analysis of resource usage data and system logs and report the success and failure of error recovery protocols. It is based on the use of TACC_Stats [17] resource usage monitor, Rationalized message logs [36] and Syslogs [40]. We implemented an approach (CORRMEXT) that has three phases. The three phases of the approach are:

- Correlating resource use counters in the resource use data to identify the strongly positive correlated resource use counters.
- Correlating message types in the system logs to identify the strongly positive correlated system events.
- Extracting the variance of the time-bins associated with the correlated resource use counters and correlated system events to identify the earliest hour of change in the system behaviour.

The CORRMEXT framework is composed of three modules. The three modules are: (1) Data Type Extraction, (2) Correlation and (3) Time-bin Extraction. When given a list of dates, CORRMEXT automatically applied the Data Type Extraction, Correlation and Time-bin Extraction modules on the resource use data and system logs. Next, we summarise the three modules within the CORRMEXT framework.

CORRMEXT: Data Type Extraction

The **Data Type Extraction** module currently processes TACC_Stats resource use data, Rationalized message logs and Syslogs. It produced a standardise data format on which standard analysis algorithms can be applied. We represented the resource use data and system logs as two data matrices. In the data matrix that represents resource use data, a resource use counter is represented by one row, a time-bin is represented by one column and the count for one resource use counter at one time-bin is represented by one cell. In the data matrix that represents the system logs, a message type is represented by one row, a time-bin is represented by one column and the count for one message type at one time-bin is represented by one cell. Our objective is to identify error propagation and recovery patterns which occur over a regular time window, not over different time windows. Therefore, we collected the resource use data and system logs using a fixed time window of one hour.

CORRMEXT: Correlation

The **Correlation** module computes: (i) the correlation coefficients between all the resource use counters and extracts a list of strongly positive correlated resource use counters for analysis, and (ii) the correlation coefficients between all the message types and extracts a list of strongly positive correlated system events for analysis. It receives the resource use counters and message types data matrices generated by the Data Type Extraction module. Currently, the Correlation module evaluated two different correlation algorithms. The correlation algorithms are Pearson correlation and Spearman-Rank correlation. We implemented Pearson correlation and use it to identify resource use counters and system events that follow a linear pattern. We implemented Spearman-Rank correlation and use it to identify resource use counters and system events that follow a monotonically increasing pattern. We combined Pearson and Spearman-Rank correlation methods to detect a gradual change or fluctuation in the relationship between a pair of resource use counters or a pair of message types. We used the following rules to interpret the strength of the correlation coefficient [1]: (i) between 0.8 to 1 as strong positive correlated, (ii) between 0.3 to 0.79 as moderate positive correlated, and (iii) between 0.1 to 0.29 as weak positive correlated.

We tested the significance of all the correlation coefficients by applying a standard technique called Fisher's z-score [67]. We defined two null hypothesis and two alternate hypothesis. The null hypotheses are: (i) a pair of resource use counters is weakly positive correlated, and (ii) a pair of message types is weakly positive correlated. The alternate hypotheses are: (i) a pair of resource use counters is strongly positive correlated, and (ii) a pair of message types is strongly positive correlated. We addressed the issue of inflation in false positive due to multiple independent tests by applying the Bonferroni correction [26].

CORRMEXT: Time-bin Extraction

At every hour, the **Time-bin Extraction** module computes the variance of the correlated resource use counters and correlated system events to identify the hour that has the highest variance. It receives the data matrices that contain the strongly positive correlated resource use counters and strongly positive correlated message types. Our objective is to identify the earliest hour of change in the system behaviour by identifying the hour containing the highest variance.

7.1.2 Error Cases Identified by CORRMEXT

We applied CORRMEXT on the resource use data and system logs collected on Ranger and Lonestar4. There are no system logs available on Stampede-1. Because of this, we applied CORRMEXT only on the resource use data collected on Stampede-1 and focus on the system logs that are available on Ranger and Lonestar4. We obtained the correlation reports that CORRMEXT has generated. We used the correlation reports to identify the error cases. We identified seven different error cases on Ranger, Lonestar4 and Stampede-1. The error cases are: (i) network data and networking software errors, (ii) filesystem, Linux process and software errors, (iii) file access and Linux process errors, (iv) Linux process errors and system memory exhaustion, (v) NUMA memory allocation and Linux software memory leaks, (vi) communication and filesystem I/O errors, and (vii) chipset and ECC memory errors. A summary of the error cases and their associated system components is given in Table 7.1.

Table 7.1: List of error cases identified on Ranger, Lonestar4 and Stampede-1.

HPC system	**Component**	**Error**
Lonestar4, Stampede-1	Infiniband network	Network data & software errors
Lonestar4, Stampede-1	Storage system & Linux processes	Filesystem, process & software errors
Ranger	Linux virtual memory & hard disk	1. File access & process errors 2. Process errors & memory exhaustion
Ranger	NUMA & memory allocation	Memory allocation & memory leaks
Ranger	Lustre filesystem I/O & Infiniband	Communication & file-system I/O errors
Ranger	Chipset & ECC memory	Chipset & memory errors

Network Data and Networking Software Errors

In this error case, we gave an example of diagnosing network problems through: (i) correlations between Infiniband and compute node network resource use counters, and (ii) correlations between DNS lookup failures and FTP software failures. The Infiniband and compute node network resource use counters record various activities on the network ranging from the amount of network data received to the number of data packets dropped and data frame errors. When data errors occur on the network, the correlations between networking resource use counters can be used to monitor the state of the Infiniband network. On Stampede-1, we found that: (i)

network data frame errors were strongly positive correlated to network data received on 26 out of 28 dates during February 2017, and (ii) network data CRC errors were strongly positive correlated to network data received on 26 out of 28 dates also during February 2017. This represented 92% of the dates that network data errors occurred. On Lonestar4, we found that: (i) network data frame errors were strongly positive correlated to network data packets received on 24 out of 26 dates during February 2013, and (ii) network data CRC errors were strongly positive correlated to network data packets received on 24 out of 26 dates also during February 2013. This represented 92% of the dates that network data errors occurred.

A DNS lookup failure occurred when a networking software attempted to identify a destination node on the network but the DNS server was incorrectly configured. On Lonestar4, we found that DNS lookup failure messages were strongly positive correlated to GSIFTP software error messages on six dates during February 2013. The correlated DNS lookup failures and GSIFTP software errors occurred on the same dates when network data errors also occurred. On three out of the six dates, the DNS lookup failure led to the GSIFTP software crash. However on all six dates, we found that the DNS lookup failure and GSIFTP software errors were weakly correlated to compute node soft lockups. Therefore, the DNS lookup failure and GSIFTP software errors did not cause a compute node to crash. This represented a recovery rate of 100%.

Filesystem, Linux Process and Software Errors

In this error case, we gave an example of diagnosing filesystem problems through: (i) correlations between harddisk, Lustre filesystem and Linux process resource use counters, and (ii) correlations between Linux process, Lustre filesystem and software errors. The harddisk, filesystem and Linux process resource use counters record various activities on the disk, filesystem and operating system ranging from the number of inodes allocated to the number of disk sectors written and Linux processes created. When a Linux process access the harddisk, the correlations between the harddisk, filesystem and Linux process resource use counters can be used to monitor the state of the storage and filesystem. On Stampede-1, we found that harddisk, filesystem and Linux process resource use counters were strongly positive correlated on 27 dates during February 2017. On Lonestar4, we found that: (i) filesystem and Linux process resource use counters were strongly positive correlated on 19 dates, (ii) filesystem and harddisk resource use counters were strongly positive correlated on 18 dates, and (iii) harddisk and Linux process resource use counters were strongly positive correlated on 24 dates.

A filesystem I/O error occurred when a harddisk degrades over time and star-

ted to fail. On Lonestar4, we found a strong positive correlation between filesystem I/O and Linux process errors and the errors were strongly positive correlated to compute node soft lockup on one out of six dates. This represented a failure rate of 17%. On five more dates, we found a strong positive correlation between filesystem I/O and Linux process errors and the errors were weakly correlated to compute node soft lockups. The correlated filesystem I/O, Linux process errors and compute node soft lockups coincided with the dates that harddisk, Lustre filesystem and Linux process resource use counters were strongly positive correlated. On five of the six dates, the filesystem I/O and Linux process errors did not cause a compute node to crash. This represented a recovery rate of 83%.

File Access and Linux Process Errors

In this error case, we gave an example of diagnosing file-access problems through: (i) correlations of harddisk and virtual memory resource use counters, and (ii) correlations of file access and Linux process errors. The harddisk and virtual memory resource use counters record various activities on the harddisk and virtual memory system ranging from the number of disk sectors written to the number of major page faults. When Linux uses the harddisk as memory due to low main memory, the correlations between harddisk and virtual memory resource use counters can be used to monitor the state of the system memory. On Ranger, we found that: (i) harddisk I/O and major page faults were strongly positive correlated on 25 dates, and (ii) file access errors were strongly correlated to compute node soft lockups on eight out of eight dates. The correlated file access errors and compute node soft lockups coincided with the dates that harddisk I/O and major page faults were strongly positive correlated. On all eight dates, file access errors led to a compute node crash. This represented a failure rate of 100%.

Linux Process Errors and System Memory Exhaustion

In this error case, we gave an example of diagnosing system memory problems through: (i) correlations of harddisk and virtual memory resource use counters, and (ii) correlations of Linux process and system memory exhaustion errors. The harddisk and virtual memory resource use counters record various activities on the harddisk and virtual memory system ranging from the number of disk sectors written to the number of major page faults. When Linux uses the harddisk as memory due to low main memory, the correlations between harddisk and virtual memory resource use counters can be used to monitor the state of the system memory. On Ranger, we found that: (i) harddisk I/O and major page faults were strongly positive correlated on 25 dates, and (ii) Linux process and system memory exhaustion errors

were strongly correlated on two dates. The correlated Linux process and system memory exhaustion errors were also strongly positive correlated to compute node soft lockups on the two dates. All the dates of the correlated Linux process, system memory exhaustion errors and compute node soft lockups coincided with the dates that harddisk I/O and major page faults were correlated. On two out of the two dates, Linux process and system memory exhaustion errors led to a compute node crash. This represented a failure rate of 100%.

NUMA Memory Allocation and Linux Software Memory Leaks

In this error case, we gave an example of diagnosing memory allocation and application memory leak through: (i) correlations of NUMA and Linux process resource use counters, and (ii) correlations of Linux software memory leaks. The NUMA and Linux process resource use counters record node memory allocation activities based on a Linux process request. When a node runs out of memory, the correlations between NUMA and Linux process resource use counters can be used to monitor the state of memory allocation in the system. On Ranger, we found that: (i) NUMA and Linux process resource use counters were strongly positive correlated on 25 dates, and (ii) segmentation fault and general protection error messages were strongly positive correlated on 10 dates. All the dates of the correlated segmentation fault and general protection errors coincided with the dates that NUMA and Linux process resource use counters were correlated. We found that the correlated segmentation fault and general protection errors were strongly positive correlated to compute node soft lockups on 2 of the 10 dates. On the two dates, segmentation fault and general protection errors led to a compute node crash. This represented a failure rate of 20%. On 8 of the 10 dates, the correlated segmentation fault and general protection errors were weakly correlated to compute node soft lockups – segmentation fault and general protection errors did not lead to a compute node crash. This represented a recovery rate of 80%.

Communication and Filesystem I/O Errors

In this error case, we gave an example of diagnosing communication and filesystem errors through: (i) correlations of Lustre filesystem I/O and Infiniband resource use counters, and (ii) correlations of Lustre filesystem and communication errors. The Lustre filesystem I/O and Infiniband resource use counters record filesystem and network traffic activities. When the network and filesystem are heavily used, the correlations between network data error and filesystem I/O resource use counters can be used to monitor the state of the networked filesystem. On Ranger, we found that: (i) Infiniband network packet drop and Lustre I/O resource use counters were

strongly positive correlated on 24 dates, and (ii) communication and filesystem errors were strongly positive correlated on 11 dates. All the dates of correlated communication and filesystem errors coincided with the dates of correlated network packet drop and Lustre I/O resource use counters. On 2 of the 11 dates, the correlated communication and filesystem errors were strongly positive correlated to compute node soft lockups. On the 2 dates, communication and filesystem errors led to a compute node crash. This represented a failure rate of 18%. On 9 of the 11 dates, the correlated communication and filesystem errors were weakly correlated to compute node soft lockups. On the 9 dates, communication and filesystem errors did not lead to a compute node crash. This represented a recovery rate of 81%.

Chipset and ECC Memory Errors

In this error case, we gave an example of diagnosing memory errors through: (i) correlations of CPU and memory resource use counters, and (ii) correlations of chipset and ECC errors. The CPU and memory resource use counters record various activities ranging from user and system CPU usage to active and inactive memory usage. When the CPU and memory are heavily used, the correlations between CPU and memory resource use counters can be used to monitor the state of the CPU and memory system. On Ranger, we found that: (i) user and system CPU and active and inactive memory resource use counters were strongly positive correlated on 13 dates, and (ii) chipset and ECC errors were strongly positive correlated on 26 dates. All the dates of correlated CPU and memory resource use counters coincided with the dates of correlated chipset and ECC errors. On all the 26 dates, chipset and ECC errors were weakly correlated to compute node soft lockups – chipset and ECC errors did not lead to a compute node crash. This represented a recovery rate of 100%.

7.2 EXERMEST Failure Diagnosis Framework

In this section, we summarise the EXERMEST failure diagnosis framework and describe the list of error cases.

7.2.1 Introduction

We developed the EXERMEST framework to identify the significant resource use counters and system events associated with rare error cases. By significant, we mean the resource use counters and system events assigned the highest scores by the feature extractors. EXERMEST used the resource use data matrices and message types data matrices. It is based on the data matrices that are generated from

TACC_Stats [17] resource usage monitor, Rationalized message logs [36] and Syslogs [40]. EXERMEST evaluated multiple feature extraction methods. We implemented an approach (EXERMEST) that has two phases. The two phases of the approach are:

- Extracting the significant resource use counters and system events by applying different feature extraction methods.
- Correlating the significant resource use counters to other resource use counters and correlating the significant system events to other system events by applying different correlation algorithms.

The EXERMEST framework is composed of two modules. The two modules are: (i) Feature Extraction, and (ii) Correlation. When given a list of dates, EXERMEST automatically applied the Feature Extraction and Correlation modules on the resource use data and message types data matrices. Next, we summarise the two modules within the EXERMEST framework.

EXERMEST: Feature Extraction

The **Feature Extraction** module receives as its input, a resource use counters data matrix and message types data matrix. In the resource use counters data matrix a resource use counter is represented by one row, a time-bin is represented by one column and the count for the resource use counter at a time-bin is represented by one cell in the data matrix. In the message types data matrix, a message type is represented by one row, a time-bin is represented by one column and the count for the message type at a time-bin is represented by one cell in the data matrix. The Feature Extraction module currently process three different time-bins of data matrices. The time-bins are 10 minutes, 30 minutes and 1 hour.

Currently, our Feature Extraction module evaluated three different feature extraction methods. The feature extraction methods are: (i) Principal Component Analysis (PCA), (ii) Independent Component Analysis (ICA), and (iii) Non-linear Principal Component Analysis (NLPCA). Our objective for applying different feature extraction methods is to identify different patterns of significant resource use counters and message types. We used PCA and ICA to extract the resource use counters and message types that follow a linear uncorrelated pattern. We used NLPCA to extract the resource use counters and message types that follow a non-linear uncorrelated pattern.

We implemented a process to extract the significant resource use counters and message types. The process is as follows: First, we extracted the scores of the supplied data on the first principal component and obtain three sets of scores by

PCA, ICA and NLPCA. Second, we sorted the scores by PCA, ICA and NLPCA in descending order. Third, we extracted the top 5% of the scores by PCA, ICA and NLPCA and obtain three lists of significant resource use counters and three lists of significant message types.

EXERMEST: Correlation

The **Correlation** module received as its input the data matrices as follows: (i) the resource use data matrix and a smaller data matrix that contains the significant resource use counters, and (ii) the message types data matrix and a smaller data matrix that contains the significant message types. Our Correlation module computed:

- The correlation coefficients between the significant resource use counters and all the other resource use counters and extracts a smaller set of resource use counters for analysis.
- The correlation coefficients between the significant message types and all the other message types and extracts a smaller set of message types for analysis.

Currently, the Correlation module applies two different correlation algorithms. The correlation algorithms are Pearson correlation and Spearman-Rank correlation. We used Pearson correlation to identify pairs of resource use counters and pairs of message types that has a linear relationship. We used Spearman-Rank correlation to identify pairs of resource use counters and pairs of message types that has a monotonically increasing relationship. We used the following rules to interpret the strength of the correlation coefficient [1]: (i) between 0.8 to 1 as strong positive correlated, (ii) between 0.3 to 0.79 as moderate positive correlated, and (iii) between 0.1 to 0.29 as weak positive correlated.

We applied a standard technique called Fisher's z-score [67] to test the significance of all the correlation coefficients. We tested the correlation coefficients by defining two null hypothesis and two alternate hypothesis. The null hypotheses are: (i) a pair of resource use counters is weakly positive correlated, and (ii) a pair of message types is weakly positive correlated. The alternate hypotheses are: (i) a pair of resource use counters is strongly positive correlated, and (ii) a pair of message types is strongly positive correlated. We applied the Bonferroni correction [26] to address the inflation in false positive due to multiple independent tests. We addressed the inflation in false positive by multiplying the P-value by the number of tests.

7.2.2 Error Cases Identified by EXERMEST

We applied EXERMEST on the resource use data and system logs collected on Ranger and Lonestar4. We obtained the features and correlation reports generated by EXERMEST. We used the reports to identify the rare error cases. All the error cases are different. The error cases are: (i) network packet drop and Lustre I/O errors, (ii) CPU I/O bottleneck and Lustre client eviction, (iii) virtual memory and harddisk I/O errors, and (iv) data and network synchronisation errors. A summary of the error cases and their associated system components is given in Table 7.2.

Table 7.2: List of rare error cases on Ranger and Lonestar4.

Ranger		
Components	**Error**	**Date**
Infiniband & Lustre network, Lustre filesystem	Network packet drops & Lustre I/O	July 5
CPU, harddisk & Lustre filesystem	CPU I/O bottleneck & Lustre client eviction	July 15
Memory management, harddisk & Lustre	Virtual memory & harddisk I/O	July 25
Lonestar4		
Components	**Error**	**Date**
Memory, Linux threads & Lustre network	Data & network synchronisation	March 1

Network Packet Drop and Lustre I/O Error

In this error case, we gave an example of diagnosing buffer overflow on a network card through: (i) correlation of significant Infiniband network packet drop to Lustre network packet drop and Lustre I/O resource use counters and correlation of significant Lustre write bytes to Lustre network packet drop and Lustre I/O resource use counters, and (ii) correlation of significant Linux kernel I/O function calls to Lustre I/O and Lustre communication errors. When network data transmission error occurred due to a network card buffer overflow, the correlated Infiniband, Lustre network and Lustre I/O resource use counters can be used to monitor the state of the Infiniband, Lustre network and filesystem.

We identified Infiniband network packet drop and Lustre write bytes as significant resource use counters on Ranger. We found that: (i) the significant Infiniband network packet drop resource use counter was only identified by NLPCA, and (ii) the significant Lustre write bytes resource use counter was only identified by PCA and ICA. Further, we found that the significant resource use counters were only identified on the data matrix of 10 minute time-bins. We identified two Linux kernel

I/O function calls as significant system events. We found that the two Linux kernel I/O function calls were only identified by NLPCA. Further, we found that both function calls were only identified on the data matrix of 1 hour time-bins. Our results showed that: (i) we require different feature extractors for identifying the significant resource use counters, and (ii) we require time-bins of different granularities for identifying the significant resource use counters and system events.

CPU I/O Bottleneck and Lustre Client Eviction Error

In this error case, we gave an example of diagnosing bottlenecks through: (i) correlations of significant CPU I/O wait to CPU idle, harddisk sectors merged and NUMA resource use counters and correlations of significant Lustre I/O call to CPU idle, harddisk sectors merged and NUMA resource use counters, and (ii) correlations of significant Lustre communication errors to Lustre client eviction events. When a Lustre filesystem client is evicted due to communication errors, the correlated CPU I/O wait to CPU idle, harddisk sectors merged and NUMA resource use counters and correlated Lustre I/O function call to CPU idle, harddisk sectors merged and NUMA resource use counters can be used to monitor bottlenecks on the CPU, harddisk and Lustre filesystem.

We identified CPU I/O wait and Lustre I/O function call as significant resource use counters on Ranger. We found that: (i) both significant resource use counters were only identified by NLPCA, and (ii) both significant resource use counters were only identified on the data matrix of 10 minute time-bins. We identified two Lustre filesystem communication error messages as significant events. We found that: (i) both significant error messages were only identified by ICA, and (ii) both significant error messages were only identified on the data matrix of 10 minute time-bins. Our results showed that we require different feature extractors for identifying the significant resource use counters and error messages.

Virtual Memory and Harddisk I/O Error

In this error case, we gave an example of diagnosing a virtual memory allocation error through: (i) correlations of significant page fault to harddisk sectors written, memory writeback and Lustre write bytes resource use counters and correlations of Lustre I/O function call to harddisk sectors written, memory writeback and Lustre write bytes resource use counters, and (ii) correlations of significant Linux kernel memory management function calls to Linux page fault and soft lockup events. When the Linux operating system crashed due to insufficient main memory and virtual memory allocation, the correlated page fault, harddisk sectors written, memory writeback and Lustre write bytes resource use counters and correlated Lustre I/O function

call, harddisk sectors written, memory writeback and Lustre write bytes resource use counters can be used to monitor the state of Linux memory management.

We identified virtual memory page fault and Lustre I/O function call as significant resource use counters on Ranger. We found that: (i) the virtual memory page fault resource use counter was only identified by ICA, and (ii) the Lustre I/O function call resource use counter was only identified by NLPCA. Further, we found both resource use counters only in the data matrix of 10 minute time-bins. We identified two Linux memory management function calls as significant system events. We found that: (i) both function call messages were only identified by NLPCA, and (ii) both function call messages were only identified on the data matrix of 1 hour time-bins. Our results showed that: (i) we require different feature extractors for identifying the significant resource use counters, and (ii) we require time-bins of different granularities for identifying the significant resource use counters and system events.

Data and Network Synchronisation Error

In this error case, we gave an example of diagnosing a synchronisation error through: (i) correlations of significant Linux threads count to memory state resource use counters and correlations of significant out-of-sync memory data to memory state resource use counters, and (ii) correlations of significant data synchronisation errors to communication errors. When data and communication errors occur due to synchronisation issues on the network and Linux operating system, the correlated Linux threads count and memory state resource use counters can be used to monitor the state of data synchronisation on the Linux operating system.

We identified Linux threads count and memory data out-of-sync as significant resource use counters on Lonestar4. We found that the significant Linux threads count resource use counter was only identified by NLPCA. Further, it was only identified on the data matrix of 10 minute time-bins. We found that the memory data out-of-sync resource use counter was identified by ICA and NLPCA. Further, the significant memory data out-of-sync resource use counter was identified: (i) only on the data matrix of 10 minute time-bins by NLPCA, and (ii) only on the data matrix of 1 hour time-bins by ICA. We identified two synchronisation error messages as significant system events. We found that one of the significant synchronisation error message was only identified by NLPCA. Further, it was identified only on the data matrix of 30 minute time-bins. We found that the second significant synchronisation error message was only identified by ICA. Further, it was identified on the data matrices of 10 minute time-bins and 1 hour time-bins. Our results showed that: (i) we require different feature extractors for identifying the significant resource use

counters and error messages, and (ii) we require time-bins of different granularities for identifying the significant resource use counters and error messages.

7.3 Similarities and Differences Between CORRMEXT and EXERMEST

In this section, we discuss the similarities and differences between the CORRMEXT and EXERMEST failure diagnosis frameworks. We begin by comparing the methods within the CORRMEXT and EXERMEST frameworks. Then, we highlight the types of errors CORRMEXT and EXERMEST were designed to identify. Then, we highlight the similarities of CORRMEXT and EXERMEST which can help integrate both workflows.

7.3.1 Features of CORRMEXT and EXERMEST

The CORRMEXT framework integrated the Data Type Extraction, Correlation and Time-bin Extraction modules. The EXERMEST framework integrated the Feature Extraction and Correlation modules. A comparison of the functions of CORRMEXT and EXERMEST is given in Table 7.3.

Table 7.3: Comparing features of the CORRMEXT and EXERMEST frameworks.

	Multiple time-bins	**Feature selection**	**Feature extraction**	**Raw data**	**Data matrix**
CORRMEXT	No	Yes	No	Yes	Yes
EXERMEST	Yes	Yes	Yes	No	Yes

We summarise the functions of CORRMEXT that are different to EXERMEST:

- It uses the raw resource use data and system logs as input.
- It organises the resource use data and system logs into time-bins of 1 hour.
- It applies supervised methods to select the features to be used for detailed diagnosis. The methods are Pearson correlation and Spearman-Rank correlation.

We summarise the functions of EXERMEST that are different to CORRMEXT:

- It uses the resource use data matrices and message types data matrices as input.

- It processes resource use data matrices and message types data matrices of multiple granularities of time-bins.
- It applies unsupervised methods to extract the features for an initial diagnosis. The methods are PCA, ICA and NLPCA.

7.3.2 Frequent and Rare Error Cases

The CORRMEXT framework identifies error cases that occur frequently and reports the success and failure of error recovery protocols. The EXERMEST framework identifies the significant system events and resource use counters to diagnose error cases that are rare. A summary of the types of error cases is given in Table 7.4.

Table 7.4: Summary of error cases diagnosed using CORRMEXT and EXERMEST.

	Multiple components	**Rare errors**	**Frequent errors**
CORRMEXT	Yes	No	Yes
EXERMEST	Yes	Yes	No

From Table 7.4, we observe that the type of error cases CORRMEXT and EXERMEST identified are different in the following way: (i) the error cases CORRMEXT identifies are frequently occurring ones, (ii) the error cases EXERMEST identifies are rare. Therefore, we can use CORRMEXT to identify error cases that occur frequently and use EXERMEST to identify error cases that are rare. Both EXERMEST and CORRMEXT identify errors that occur on multiple system components.

7.3.3 Integrating EXERMEST and CORRMEXT

From Table 7.3, we observe that CORRMEXT and EXERMEST shared two functions. The functions are: (i) using correlation to select features for detailed diagnosis, (ii) using resource use data matrices and message types data matrices to identify the features which are relevant to the diagnosis of an error. From Table 7.4, we observe that CORRMEXT and EXERMEST identified errors that occur on multiple system components. Based on the similarities, a framework which integrates the CORRMEXT and EXERMEST workflows can be developed. We are implementing a framework to bring the workflows together.

7.4 Summary

We presented a comparative analysis of the CORRMEXT and EXERMEST failure diagnosis frameworks. We summarised the Data Type Extraction, Correlation and

Time-bin Extraction modules used within CORRMEXT and described the list of error cases. We summarised the Feature Extraction and Correlation modules used within EXERMEST and described the list of error cases. We discussed the features of CORRMEXT and EXERMEST and showed that CORRMEXT and EXERMEST complemented each other through their diagnostics functions.

Chapter 8

Summary and Future Research

Analysing failures to improve the reliability of HPC systems is important. The system logs and resource use data that HPC systems generate are a useful source of information, however the large amount of data presents a significant challenge for systems diagnosis. In this thesis, we developed two failure diagnosis frameworks. We implemented the Data Type Extraction, Feature Extraction, Correlation and Time-bin Extraction modules. We integrated the Data Type Extraction, Correlation and Time-bin Extraction modules into the framework called CORRMEXT. It identified frequently occurring error cases and reported the success and failure of error recovery protocols. We integrated the Feature Extraction and Correlation modules into the framework called EXERMEST. It extracted the significant system events and resource use counters associated with rare error cases.

We structure the remainder of this chapter as follows: In Section 8.1, we summarise the contribution of Chapters 2, 3, 4, 5, 6 and 7. In Section 8.2, we provide a number of suggestions for future research.

8.1 Summary of Chapters

The introductory chapters in this thesis are: (i) Introduction (Chapter 1), (ii) Review of cluster log-files data processing tools (Chapter 2), and (iii) Define the research problem, describe the HPC systems and log-data (Chapter 3). The contribution chapters in this thesis are: (i) The CORRMEXT framework (Chapters 4 and 5), and (ii) The EXERMEST framework (Chapter 6).

In Chapter 2, we presented a detailed survey of system log-file processing tools. We showed that most of the tools were comprised of two main activities. The first activity tokenised system log messages by extracting sequences of English-only words in the free form text of the textual message logs. The second activity applied a method to measure the similarity between system log messages. Further, we

showed that the method implemented by the tools can be grouped into five different categories.

In Chapter 3, we described the system model, fault model and system issues, introduced the Ranger, Lonestar4 and Stampede-1 HPC systems operated by the Texas Advanced Computing Center, described the TACC_Stats resource use data, Rationalized message logs and Syslogs, and gave the implementation details for the Resource Use Data and Message Types Data preprocessing modules.

In Chapter 4, we presented the CORRMEXT framework that processed both the resource use data and system logs to identify: (i) frequently occurring error cases, and (ii) report the success and failure of error recovery protocols. We applied CORRMEXT on the TACC_Stats resource use data and Rationalized message logs on the Ranger HPC system. CORRMEXT diagnosed five error cases and extracted the variance in the times of the correlated resource use counter groups and correlated error groups to identify the earliest occurrences of the problem. To ensure diagnostics accuracy, CORRMEXT used the Bonferroni correction and showed that all the correlations are significant. The main technical contribution of the chapter is a new failure diagnostics framework that integrated resource use data and system log-preprocessing, multiple correlation methods and time-bins extraction.

In Chapter 5, we applied CORRMEXT on multiple HPC systems. CORRMEXT generated the analyses focused on the correlated groups of resource use counters and correlated groups of errors. It confirmed the findings reported in Chapter 4 and identified three different error cases and one resource usage activity case. To ensure diagnostics accuracy, we showed that all the correlations would not be observed under the null hypothesis.

In Chapter 6, we presented the EXERMEST framework for diagnosing rare error cases on multiple HPC systems. EXERMEST used the resource use data matrices and message types data matrices in its analyses. It evaluated three different feature extraction methods. EXERMEST identified four different error cases on two HPC systems. EXERMEST used the Bonferroni correction to ensure the accuracy of the diagnoses and showed that all the correlations would not be observed under the null hypothesis. The main technical contribution of the chapter is a new workflow that integrated multiple feature extraction methods and multiple correlation methods.

In Chapter 7, we presented a comparative analysis of the CORRMEXT and EXERMEST failure diagnosis frameworks. We summarised the Data Type Extraction, Correlation and Time-bin Extraction modules used within CORRMEXT and provided the list of error cases. We summarised the Feature Extraction and Correlation modules used within EXERMEST and provided the list of error cases. We discussed the various functions in CORRMEXT and EXERMEST and showed

that they complement one another by identifying their similarities.

8.2 Future Research

In this section, we provide a number of suggestions for future research.

8.2.1 Preventive Maintenance

The groups of correlated resource use counters and groups of correlated errors (presented in Chapters 4 and 5) raise an interesting point for discussion on further work. A detailed analysis of spatial failures on a large HPC system was presented in [29]. The authors used the analyses to improve the performance of applications running on the Titan supercomputer at Oak Ridge National Laboratory. They proposed a novel scheme that exploited the spatial characteristics of system failures. In contrast to the analyses presented in [29], we argue that the CORRMEXT framework focuses on the problem specification described in Section 4.2.2, i.e., *we seek to correlate groups of resource use counters and correlate groups of errors by time-bins.* We know that network data and networking software errors, file access errors and process messages, process errors and system memory exhaustion, and filesystem, Linux process and software errors can be identified using hour-based correlation analysis. A potential thread for future research is to integrate location and time-based analysis approaches to relate system processes to new error cases. This will lead to an interesting application where the diagnoses can be used as part of a preventive maintenance regime. Here, the goal is to identify small problems and fix them before they become a major problem.

8.2.2 Extension to Distributed Systems

The HPC systems presented in Chapter 3 raise an interesting point for discussion on further work. The CORRMEXT framework does not process system logs generated on distributed systems but CORRMEXT's data type extractor was designed to work on system logs that contain only three fields (timestamp, node and message). We argue that this pertains to the case study systems, i.e., *we identify the error cases on HPC systems.* Having said that, we can apply CORRMEXT's data type extractor to process system logs generated on distributed systems, for example: Hadoop system logs. We can integrate distributed log-analysis frameworks, for example: DILAF [2] with CORRMEXT. A potential thread for future research is to extend CORRMEXT to diagnose errors on distributed systems.

8.2.3 Failure Prediction

The error cases presented in Chapters 4, 5 and 6 raise an interesting point for discussion on further work. The EXERMEST and CORRMEXT frameworks do not infer actual root cause of system failures but it is done with the aid of a system administrator. We argue that this pertains to the fault model described in Section 6.2.1, i.e., *we identify errors without prior knowledge of fault models*. There is no prediction model. Therefore, we focus on system diagnosis. Having said that, we can: (i) use the error cases identified by EXERMEST and (ii) use the frequently occurring error cases identified by CORRMEXT to predict when those errors are likely to occur in the future. We can integrate online failure prediction frameworks, for example: the reference in [52], failure prediction techniques described in [59] with EXERMEST and CORRMEXT. Integrating EXERMEST and CORRMEXT with an online failure prediction framework and error detection method, for example: CRUDE [34] will expand it into the error prediction and detection phases. A potential thread for future research is to develop an unsupervised approach to identify which resource use counter and system event are good predictors for the error cases.

8.2.4 Feature Selection and Optimisation

The Correlation module implemented in CORRMEXT raises an interesting point for further work. CORRMEXT generated the correlation reports which is manually scanned to extract the resource use counters and system errors for diagnosing an error case. We argue that this pertains to the problem specification described in Section 4.2.2, i.e., *we seek to correlate groups of resource use counters and correlate groups of errors by time-bins*. Having said that, we can integrate feature extraction with correlation, for example: in EXERMEST (see Chapter 6). We can also integrate correlation-based feature selection techniques, for example: the reference in [43] with CORRMEXT. A potential thread for future research is to develop a correlation-based approach to automate selection of a good subset of features whilst minimising the error rate.

Bibliography

[1] Alan Agresti and Christine Franklin. *Statistics: The Art and Science of Learning From Data*. Prentice Hall International, 2009. ISBN 978-0135131992.

[2] Merve Astekin, Harun Zengin, and Hasan Szer. DILAF: A framework for distributed analysis of large-scale system logs for anomaly detection. *Software: Practice and Experience*, 49(2):153–170, 2019. doi: 10.1002/spe.2653.

[3] Algirdas Avizienis, Jean-Claude Lapire, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[4] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448 – 461, 1973. doi: https://doi.org/10.1016/S0022-0000(73)80033-9.

[5] Greg Bronevetsky, Ignacio Laguna, Bronis R de Supinski, and Saurabh Bagchi. Automatic fault characterization via abnormality-enhanced classification. In *Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12, 2012. doi: 10.1109/DSN.2012.6263926.

[6] Edward Chuah and et. al. Enabling online resource use and system log-analysis for HPC systems vulnerability diagnosis. In *Under preparation for submission in 2020*, 2020.

[7] Edward Chuah, Gary Lee, William-Chandra Tjhi, Shyh-Hao Kuo, Terence Hung, John Hammond, Tommy Minyard, and James C. Browne. Establishing hypothesis for recurrent system failures from cluster log files. In *Proceedings of IEEE DASC*, pages 1–8, Dec 12-14 2011.

[8] Edward Chuah, Arshad Jhumka, Sai Narasimharmuthy, John Hammond, James C. Browne, and Bill Barth. Linking resource usage anomalies with system failures from cluster log data. In *Proceedings of IEEE International*

Symposium on Reliable Distributed Systems (SRDS), pages 111–120, 2013. doi: 10.1109/SRDS.2013.20.

[9] Edward Chuah, Arshad Jhumka, James C. Browne, Nentawe Gurumdimma, Sai Narasimharmuthy, and Bill Barth. Using message logs and resource use data for cluster failure diagnosis. In *Proceedings of IEEE International Conference on High Performance Computing, Data and Analytics (HiPC)*, pages 232–241, 2016. doi: 10.1109/HiPC.2016.035.

[10] Edward Chuah, Arshad Jhumka, Samantha Alt, Theo Damoulas, Nentawe Gurumdimma, Marie-Christine Sawley, William L. Barth, Tommy Minyard, and James C. Browne. Enabling dependability-driven resource use and message-log analysis for cluster system diagnosis. In *Proceedings of IEEE International Conference on High Performance Computing, Data and Analytics (HiPC)*, pages 317–327, 2017. doi: 10.1109/HiPC.2017.00044.

[11] Edward Chuah, Arshad Jhumka, Samantha Alt, Daniel Balouek-Thomert, James C. Browne, and Manish Parashar. Towards comprehensive dependability-driven resource use and message log-analysis for HPC systems diagnosis. *Journal of Parallel and Distributed Computing*, 132:95–112, 2019. doi: https://doi.org/10.1016/j.jpdc.2019.05.013.

[12] Edward Chuah, Arshad Jhumka, Samantha Alt, J.J Villalobos, Joshua B. Fryman, William L. Barth, and Manish Parashar. Using resource use data and system logs for HPC system error propagation and recovery diagnosis. In *Proceedings of IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 1–10, 2019.

[13] Christophe Croux, Peter Filzmoser, and M. Rosario Oliveira. Algorithms for projection-pursuit robust principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 87(2):218–225, 2007.

[14] Anwesha Das, Frank Mueller, Paul Hargrove, Eric Roman, and Scott Baden. Doomsday: Predicting which node will fail when on supercomputers. In *IEEE/ACM Supercomputing (SC)*, 2018.

[15] Anwesha Das, Frank Mueller, Charles Siegel, and Abhinav Vishnu. Desh: Deep learning for system health prediction of lead times to failure in HPC. In *ACM HPDC*, 2018.

[16] S. Di, R. Gupta, M. Snir, E. Pershey, and F. Cappello. Logaider: A tool for mining potential correlations of HPC log events. In *IEEE Cluster, Cloud and Grid Computing (CCGRID)*, pages 442–451, May 2017.

[17] R. Todd Evans, James C. Browne, and William L. Barth. Understanding application and system performance through system-wide monitoring. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1702–1710, 2016. doi: 10.1109/IPDPSW.2016.145.

[18] Ilenia Fronza, Alberto Sillitti, Giancarlo Succi, Mikko Terho, and Jelena Vlasenko. Failure prediction based on log files using random indexing and support vector machines. *Journal of Systems and Software*, 86(1):2 – 11, 2013. doi: https://doi.org/10.1016/j.jss.2012.06.025.

[19] Q. Fu, J. G. Lou, Y. Wang, and J. Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 Ninth IEEE International Conference on Data Mining*, pages 149–158, Dec 2009. doi: 10.1109/ICDM.2009.60.

[20] Song Fu and Cheng-Zhong Xu. Exploring event correlation for failure prediction in coalitions of clusters. In *Proceedings of ACM/IEEE Supercomputing*, number 41, 2007.

[21] Xiaoyu Fu, Rui Ren, Jianfeng Zhan, Wei Zhou, Zhen Jia, and Gang Lu. Logmaster: Mining event correlations in logs of large-scale cluster systems. In *Proceedings of IEEE International Symposium on Reliable Distributed Systems (SRDS)*, pages 71–80, 2012. doi: 10.1109/SRDS.2012.40.

[22] Errin W. Fulp, Glenn A. Fink, and Jereme N. Haack. Predicting computer system failures using support vector machines. In *Proceedings of 1st USENIX Workshop on the Analysis of System Logs*, 2008. URL `http://dl.acm.org/citation.cfm?id=1855886.1855891`.

[23] Ana Gainaru, Franck Cappello, Stefan Trausan-Matu, and Bill Kramer. Event log mining tool for large scale HPC systems. In *Proceedings of Euro-Par*, pages 52–64, 2011.

[24] Ana Gainaru, Franck Cappello, and William Kramer. Taming of the shrew: Modeling the normal and faulty behaviour of large-scale HPC systems. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1168–1179, 2012. doi: 10.1109/IPDPS.2012.107.

[25] Diego Galar and Uday Kumar. Chapter 5 - diagnosis. *eMaintenance*, pages 235 – 310, 2017.

[26] Jelle J. Goeman and Aldo Solari. Multiple hypothesis testing in genomics. *Statistics in Medicine*, 33(11):1946–1978, 2014. doi: 10.1002/sim.6082.

[27] Qiang Guan and Song Fu. Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures. In *32nd IEEE International Symposium on Reliable Distributed Systems (SRDS)*, pages 205–214, 2013. doi: 10.1109/SRDS.2013.29.

[28] Qiang Guan, Derek Smith, and Song Fu. Anomaly detection in large-scale coalition clusters for dependability assurance. In *Proceedings of IEEE International Conference on High Performance Computing (HiPC)*, pages 1–10, 2010. doi: 10.1109/HIPC.2010.5713169.

[29] S. Gupta, D. Tiwari, C. Jantzi, J. Rogers, and D. Maxwell. Understanding and exploiting spatial properties of system failures on extreme-scale HPC systems. In *Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 37–44, 2015. doi: 10.1109/DSN.2015.52.

[30] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. Failures in large scale systems: Long-term measurement, analysis, and implications. In *Proceedings of IEEE/ACM Supercomputing (SC)*, pages 44:1–44:12, 2017. doi: 10.1145/3126908.3126937.

[31] Nentawe Gurumdimma and Arshad Jhumka. Detection of recovery patterns in cluster system using resource usage data. In *Proceedings of IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 58–67, 2017. doi: 10.1109/PRDC.2017.17.

[32] Nentawe Gurumdimma, Arshad Jhumka, Maria Liakata, Edward Chuah, and Jamee C. Browne. On handling redundancy for failure log analysis of cluster systems. In *Proceedings of DEPEND*, pages 1–8, 2015.

[33] Nentawe Gurumdimma, Arshad Jhumka, Maria Liakata, Edward Chuah, and James C. Browne. Towards increasing the error handling time window in large-scale distributed systems using console and resource usage logs. In *Proceedings of IEEE Trustcom/BigDataSE/ISPA*, pages 61–68, 2015. doi: 10.1109/Trustcom.2015.613.

[34] Nentawe Gurumdimma, Arshad Jhumka, Maria Liakata, Edward Chuah, and James C. Browne. Crude: Combining resource usage data and error logs for accurate error detection in large-scale distributed systems. In *Proceedings of IEEE International Symposium on Reliable Distributed Systems (SRDS)*, pages 51–60, 2016. doi: 10.1109/SRDS.2016.017.

[35] John Hammond. Tacc_stats: I/o performance monitoring for the intransigent. In *Invited Keynote for the 3rd IASDS Workshop*, pages 1–29, 2011.

[36] John L. Hammond, Tommy Minyard, and Jim Browne. End-to-end framework for fault management for open source clusters: Ranger. In *Proceedings of ACM TeraGrid Conference*, number 9, 2010. doi: 10.1145/1838574.1838583.

[37] Stephen E. Hansen and E. Todd Atkins. Automated system monitoring and notification with swatch. In *USENIX LISA*, pages 101–108, 1993.

[38] Huan Liu and Lei Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502, 2005.

[39] Aapo Hyvarinen and Erkki Oja. Independent component analysis: Algorithms and applications. *Neural Networks*, 13(4-5):411–430, 2000.

[40] IEEE. *IEEE Std 1003.1-2001 Standard for Information Technology — Portable Operating System Interface (POSIX) Rationale (Informative)*. IEEE Standards, 2001. ISBN 1-85912-247-7 (UK), 1-931624-07-0 (US), 0-7381-3048-6 (print), 0-7381-3010-9 (PDF), 0-7381-3129-6 (CD-ROM).

[41] Soila P Kavulya, Scott Daniels, Kaustubh Joshi, Matti Hiltunen, Rajeev Gandhi, and Priya Narasimhan. Draco: Statistical diagnosis of chronic problems in large distributed systems. In *Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12, 2012. doi: 10.1109/DSN.2012.6263927.

[42] Agnan Kessy, Alex Lewin, and Korbinian Strimmer. Optimal whitening and decorrelation. *The American Statistician*, 0(0):1–6, 2018.

[43] Madhumathi R. Kowshalya, A.M. and N. Gopika. Correlation based feature selection algorithms for varying datasets of different dimensionality. *Wireless Personal Communications*, (108):19771993, 2019. doi: https://doi.org/10.1007/s11277-019-06504-w.

[44] Zhiling Lan, Ziming Zheng, and Yawei Li. Toward automated anomaly identification in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 21(2):174–187, 2010.

[45] Yinglung Liang, Yanyong Zhang, Morris Jette, Anand Sivasubramaniam, and Ramendra Sahoo. Bluegene/l failure analysis and prediction models. In *Proceedings of IEEE/IFIP DSN*, pages 425–434, 2006.

[46] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. Failure prediction in ibm bluegene/l event logs. In *Proceedings of IEEE International*

Conference on Data Mining (ICDM), pages 583–588, 2007. doi: 10.1109/ICDM.2007.46.

[47] Tao Li Liang Tang and Chang-Shing Perng. Logsig: Generating system events from raw textual logs. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, pages 785–794, 2011.

[48] Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. Clustering event logs using iterative partitioning. In *Proceedings of ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 1255–1264, 2009. doi: 10.1145/1557019.1557154.

[49] Ann Gentile Matthew Wong Narate Taerat, Jim Brandt and Chokchai Leangsuksun. Baler: deterministic, lossless log message clustering tool. *Computer Science: Research and Development*, 26:285–295, 2011.

[50] Adam J. Oliner, Alex Aiken, and Jon Stearley. Alert detection in system logs. In *Proceedings of IEEE International Conference on Data Mining (ICDM)*, pages 959–964, December 2008. doi: 10.1109/ICDM.2008.132.

[51] Adam J Oliner, Ashutosh V. Kulkarni, and Alex Aiken. Using correlated surprise to infer shared influence. In *Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 191–200, 2010. doi: 10.1109/DSN.2010.5544921.

[52] Alejandro Pelaez, Andres Quiroz, James C. Browne, Edward Chuah, and Manish Parashar. Online failure prediction for HPC resources using decentralized clustering. In *Proceedings of IEEE International Conference on High Performance Computing (HiPC)*, pages 1–9, 2014. doi: 10.1109/HiPC.2014.7116903.

[53] James E. Prewett. Analyzing cluster log files using logsurfer. In *4th Linux Clusters Conference CWCE*, June 2003.

[54] James E. Prewett. Listening to your cluster with logs. In *Proceedings of the 5th LCI International Conference on Linux Clusters: TheHPC Revolution*, 2004. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.563.8475`.

[55] Thomas Reidemeister, Mohammad Ahmad Munawar, Miao Jiang, and Paul A.S. Ward. Diagnosis of recurrent faults using log files. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, pages 12–23, 2009. doi: 10.1145/1723028.1723031.

[56] John P. Rouillard. Real-time log file analysis using the simple event correlator (sec). In *Proceedings of the 18th USENIX Conference on System Administration*, pages 133–150, 2004. URL `http://dl.acm.org/citation.cfm?id=1052676.1052694`.

[57] Felix Salfner and Steffen Tschirpke. Error log processing for accurate failure prediction. In *1st UNIX Workshop on the Analysis of System Logs*, December 2008.

[58] Felix Salfner, Peter Troeger, and Steffen Tschirpke. Cross-core event monitoring for processor failure prediction. In *Proceedings of IEEE International Conference on High Performance Computing Simulation*, pages 67–73, 2009. doi: 10.1109/HPCSIM.2009.5191988.

[59] Felix Salfner, Maren Lenk, and Miroslaw Malek. A survey of online failure prediction methods. *ACM Comput. Surv.*, 42(3), March 2010. ISSN 0360-0300. doi: 10.1145/1670679.1670680. URL `https://doi.org/10.1145/1670679.1670680`.

[60] Matthias Scholz, Fatma Kaplan, Charles L. Guy, Joachim Kopka, and Joachim Selbig. Non-linear PCA: a missing data approach. *Bioinformatics*, 21(20), 2005.

[61] Niyazi Sorkunlu, Varun Chandola, and Abani Patra. Tracking system behavior from resource usage data. In *Proceedings of IEEE International Conference on Cluster Computing (CLUSTER)*, pages 410–418, 2017. doi: 10.1109/CLUSTER.2017.70.

[62] Jon Stearly and Adam J. Oliner. Bad words: Finding faults in spirit's syslogs. In *Proceedings of IEEE CCGRID*, pages 765–770, 2008.

[63] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2006.

[64] Michael M. Tsao and Daniel P. Siewiorek. Trend analysis on system error files. In *Proceedings of FTCS '83*, pages 116–119, 1983.

[65] Risto Vaarandi. Sec a lightweight event correlation tool. In *Proceedings of the 2002 IEEE Workshop on IP Operations and Management (IPOM)*, pages 1–5, 2002.

[66] Risto Vaarandi. Mining event logs with slct and loghound. In *Proceedings of IEEE Network Operations and Management Symposium (NOMS)*, pages 1071–1074, 2008. doi: 10.1109/NOMS.2008.4575281.

[67] Ronald E. Walpole, Raymond H. Myers, and Sharon L. Myers. *Probability and Statistics for Engineers and Scientists*. Prentice Hall International, 1998. ISBN 978-0138402082.

[68] Guosai Wang, Lifei Zhang, and Wei Xu. What can we learn from four years of data center hardware failures? In *Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 25–36, June 2017. doi: 10.1109/DSN.2017.26.

[69] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of ACM Symposium on Operating Systems Principles (SIGOPS)*, pages 117–132, 2009. doi: 10.1145/1629575.1629587.

[70] Ziming Zheng, Zhiling Lan, Byung H. Park, and Al Geist. System log pre-processing to improve failure prediction. In *Proceedings of IEEE/IFIP DSN*, 2009.

[71] Ziming Zheng, Li Yu, Wei Tang, and Zhiling Lan. Co-analysis of RAS log and job log on BlueGene/P. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 840–851, 2011. doi: 10.1109/IPDPS.2011.83.

[72] Ziming Zheng, Li Yu, Zhiling Lan, and Terry Jones. 3-dimensional root cause diagnosis via co-analysis. In *Proceedings of ACM International Conference on Autonomic Computing (ICAC)*, pages 181–190, 2012. doi: 10.1145/2371536.2371571.